Architecture decisions

Heesch, Uwe van

RIJKSUNIVERSITEIT GRONINGEN

Architecture Decisions: The next step

Understanding, modeling, supporting and reviewing architecture decisions

Proefschrift

ter verkrijging van het doctoraat in de
Wiskunde en Natuurwetenschappen
aan de Rijksuniversiteit Groningen
op gezag van de
Rector Magnificus, dr. E. Sterken,
in het openbaar te verdedigen op
dinsdag 18 december 2012
om 11:00 uur

door

Uwe van Heesch

geboren op 27 maart 1979
te Mönchengladbach, Duitsland

Promotor:	Prof. dr. P. Avgeriou
Beoordelingscommissie:	Prof. dr. P. Kruchten Prof. dr. H. van Vliet Prof. dr. C. Wohlin

ISBN: 978-90-367-5923-6

Samenvatting

Het perspectief om te kijken naar software-architectuur als het resultaat van een set architectuurbeslissingen, wordt vandaag de dag breed gedragen onder onderzoekers. Desalniettemin, hoewel vooruitgang is geboekt in het bepalen welke elementen een architectuurbeslissing zou moeten bevatten, is er momenteel geen algemeen geaccepteerde aanpak voor het modelleren van deze beslissingen. Bestaande methodes houden bij de beschrijving van een beslissing niet met alle stakeholder-belangen rekening; ze ondersteunen het architectuurproces niet op optimale wijze en ze sluiten niet goed aan bij de rest van de architectuurdocumentatie, die doorgaans in meerdere architectural views wordt beschreven. Het doel van deze dissertatie is om genoemde problemen aan te pakken door middel van een nieuwe beslissingsmodelleringsmethode. Naast integratie in de viewpoint-gebaseerde architectuurdocumentatie, zou de modelleringsmethode architecten moeten ondersteunen bij het nemen van beslissingen en tijdens architectuurevaluaties.

Om het besluitvormingsproces te kunnen ondersteunen, moeten we eerst begrijpen hoe beslissingen in de praktijk worden genomen en welke deficiënties in het redeneringsproces bestaan. Om aan dit begrip bij te dragen, rapporteert deze dissertatie over twee surveys. Het eerste survey kijkt naar het besluitvormingsproces van laatstejaars software engineering-studenten. De resultaten van het onderzoek worden vergeleken met de architectuurliteratuur, om tekortkomingen in het redeneringsproces die ondersteund zouden moeten worden door middel van systematische besluitvormingsdocumentatie, te identificeren. Het tweede onderzoek kijkt naar het optimale besluitvormingsproces van professionele architecten, waaruit een set van redenerings-best-practices wordt gedistilleerd.

Nadat we een goed beeld hebben verkregen van het besluitvormingsproces in de praktijk, zijn we gaan kijken hoe besluitvormingsmodellering kan worden verbeterd. Uitgangspunt was het ontwikkelen van een methode om beslissingen en de motivatie daarachter vast te leggen, die weinig inspanning vraagt van de architect tijdens het ontwerp-proces. Veel software-systemen zijn ontworpen aan de hand van patterns, die veel informatie geven over de toegepaste oplossing en de motivatie daarachter in de vorm van een probleembeschrijving en de forces die de selectie van een oplossing beïnvloeden. Als een toegepast pattern kan worden geïdentificeerd in een architectuurontwerp, dan kan een groot deel van de motivatie achter een besluit uit de patternbeschrijving worden afgeleid. Deze dissertatie beschrijft een gecontroleerd experiment met mensen uit de praktijk en academici, uitgevoerd om te bekijken of een fo-

cus op software patterns tijdens het achterhalen van architectuurbeslissingen, tot een hogere kwaliteit van en kwantiteit aan gevonden beslissingen leidt, in vergelijking met dit proces zonder pattern focus. Het experiment levert statistisch significant bewijs dat een focus op patterns de kwaliteit verhoogt, terwijl geen afdoende bewijs omtrent verhoogde kwantiteit is gevonden.

Pattern-based decision recovery kan op effectieve wijze helpen om architectuurbeslissingen te achterhalen en te beschrijven, maar houdt geen rekening met een aantal andere stakeholder-belangen bij het beschrijven van beslissingen. Om met deze belangen rekening te houden, en om het modelleren van beslissingen te integreren met andere viewpoint-gebaseerde architectuurbeschrijvingen, hebben we een beschrijvingsframework voor architectuurbeslissingen ontwikkeld, conform de conventies van de internationale standaard ISO/IEC/IEEE 42010. Het framework bestaat uit vijf viewpoints, ieder gewijd aan verschillende stakeholder-belangen bij architectuurbeslissingen. De compatibiliteit met ISO/IEC/IEEE 42010 maakt het mogelijk het framework te combineren met andere viewpoint-gebaseerde architectuurbeschrijvingen.

In aanvulling op het gebruik voor documentatie van architectuurbeslissingen, onderzoekt deze dissertatie het potentieel van decision viewpoints voor het ondersteunen van ontwerpers bij het nemen van rationele beslissingen. Daartoe is een vergelijkende multiple-case study met vier groepen senior software engineering-studenten uitgevoerd. De resultaten laten zien dat studenten die decision views ontwikkelen volgens het architectuurbeslissingsframework, meer systematisch mogelijke architectuuroplossingen verkennen en evalueren dan studentgroepen die het framework niet gebruiken.

Het potentieel van decision viewpoints voor het ondersteunen van rationele beslissingen leidt tot de aanname dat het achteraf achterhalen van decision views evaluatie van hoe goed besluiten aansluiten bij relevante decision forces, ondersteunt. Dientengevolge hebben we een architectuurevaluatiemethode ontwikkeld, die architectuurbeslissingen gebruikt als primaire evaluatiedoelen. De methode, genaamd *Decision Centric Architecture Review*, brengt de motivatie achter de belangrijkste architectuurbeslissingen aan het licht en evalueert deze, daarbij alle relevante forces die van invloed zijn geweest in beschouwing nemende. Hierbij wordt gebruik gemaakt van viewpoints uit het beslissingsframework om het evaluatieproces te ondersteunen. Verschillende evaluaties aan de hand van de methode bij bedrijven in het distributed machine control system-domein hebben de toepasbaarheid van DCAR in grote industriële projecten aangetoond.

Abstract

The perspective of looking at software architecture as the result of a set of architecture decisions has gained acceptance among researchers today. Nevertheless, although notable progress has been made in defining which content architecture decisions should entail, there is currently no commonly accepted approach to architecture decision modeling. Existing approaches do not satisfy all stakeholder concerns in decision description; they do not optimally support the architecting process, and they do not integrate well with the rest of the architecture documentation, which is usually arranged in multiple architectural views. The goal of this dissertation is to address the aforementioned problems by means of a new decision modeling approach. Apart from integrating into viewpoint-based architecture documentation, the modeling approach should support architects during the process of decision-making and during architecture evaluation.

In order to support the decision-making process, we first need to understand how decisions are made in practice, and which deficiencies exist in the reasoning process. To contribute to this understanding, this dissertation reports on two surveys. The first survey explores the decision-making process of final-year software engineering students. The results of the survey are compared to the architecture literature, in order to identify shortcomings in the reasoning process that should be supported by means of systematic decision documentation. The second survey was conducted to explore the optimal decision-making process of professional architects, from which we distilled a set of reasoning best-practices.

After gaining a good understanding of the decision-making process in practice, we started investigating how decision modeling can be improved. We first thought about a method to capture decisions and the rationale behind them, that does not require much effort by the architect during the design process. Many software systems are designed using patterns, which provide rich information about the applied solution and the rationale behind the solution in the form of a problem description and the forces that influence the selection of a solution. If an applied pattern can be identified in an architectural design, then a great part of the rationale that went into the decision can be deduced from the pattern description. This dissertation describes a controlled experiment with practitioners from industry and academia, which was conducted to find out if a focus on software patterns during architecture decision recovery leads to higher quality and quantity of the recovered decisions, compared to recovery that is not focused on identifying patterns. The experiment delivers statistically-significant evidence that a focus on patterns increases the quality of recovered decisions, while no conclusive

evidence concerning the quantity of recovered decisions was found.

Pattern-based decision recovery can help to recover and describe architecture decisions effectively, but it does not satisfy many other stakeholder concerns in architecture decision description. To address these concerns, and to integrate decision modeling with other viewpoint-based architecture descriptions, we developed a description framework for architecture decisions, which follows the conventions of ISO/IEC/IEEE 42010, the international standard for (software) system's architecture description. The framework consists of five interrelated viewpoints, each of which being dedicated to satisfying different stakeholder concerns in architecture decisions. The viewpoints of the framework can be used individually, or in combination, to describe the architecture decisions made in a software project. The framework's compliance with ISO/IEC/IEEE 42010 allows to combine it with other viewpoint based architecture descriptions. The framework was validated in two empirical studies, which provide evidence for the suitability of decision viewpoints to satisfy typical stakeholder concerns in architecture decision description.

In addition to being used for documenting architecture decisions, this dissertation explores the potential of decision viewpoints for supporting designers in making rational decisions. Therefore, a comparative multiple-case study was conducted with four groups of senior software engineering students. The results show that student groups, who create views according to the architecture decision framework, explore and evaluate candidate architectural solutions more systematically than student groups who do not use the decision framework.

The potential of decision viewpoints for supporting rational decisions lead to the assumption that recovering decision views after-the-fact supports evaluating how well decisions address the relevant decision forces. As a consequence, we developed an architecture evaluation method, which uses architecture decisions as primary evaluation targets. The method, called *Decision Centric Architecture Review* (DCAR), uncovers and evaluates the rationale behind the most important architecture decisions made in a software project, considering all relevant forces that must be addressed by the decisions. It uses viewpoints from the decision framework to support the evaluation process. Multiple executions of the method in companies from the distributed machine-control system domain have shown the applicability of DCAR in large industrial projects.

Contents

Acknowledgments

Finishing a PhD thesis is not a one-man project. In the last four years, I have been supported and inspired by so many people that I can only mention some of them here.

First of all, I would like to thank my wife Esther. Without your love, commitment, and understanding, I could never have finished this dissertation. You have always been a strong partner for me, who gave me back-up and support when I needed it. I thank you for having gone this often difficult, but also great path with me, for all the sacrifices you made in the last years, for being a devoted and loving mother for our children Leni and Jonne, and for sharing your life with me. My parents Ursula and Wolfgang van Heesch passed away before I started my PhD project. I wholeheartedly thank them for everything they have done for me.

Next, I want to thank Paris Avgeriou. You are the prototype of a great PhD supervisor. In particular, I thank you for all the time you spent on our countless meetings and skype calls, for giving me direction when I needed it, for always pushing me to strive for perfection, for introducing me to the right people, for standing up for me, and for being a friend. If I ever get to supervise PhD students, I will take you as a shining example.

Especially in the beginning of my PhD research, my namesake Uwe Zdun provided great ideas, support, and feedback on my early work. It was inspiring to get to know you and to collaborate with you. I also thank my colleagues and former colleagues from the SEARCH group: Peng Liang, Neil Harisson, Trosky Callo Arias, Ahmad Waqas Kamal, Klaas-Jan Stol, and Zheng Liang. Special thanks go to Matthias Galster and Dan Tofan for reviewing many of my papers. I also thank the secretaries Desiree Hansen, Esmee Elshof, and Ineke Schelhaas for all the friendly and reliable support in organizational matters.

I am truly thankful for the support of the Fontys Hogescholen, who provided me with a stipend that allowed me to conduct research in parallel to my job as a lecturer. In particular, I thank Christiane Holz, Henk van den Heuvel, and Hans Aarts for granting me financial support. Special thanks go to each one of my colleagues and former colleagues in the software engineering study program for support and back-up: Richard van den Ham, Pieter van den Hombergh, Thijs Dorssers, Ferd van Odenhoven, Marc Dessi, Gregor Schwake, Jan Jacobs, Sander Bruinsma, Edi Klein, Cees van Tilborg, and Jeu van Loon. Richard van den Ham deserves special acknowledgements for translating

the abstract of this dissertation.

I would also like to thank the members of my reading committee, Prof. dr. Philippe Kruchten, Prof. dr. Hans van Vliet, and Prof. dr. Claes Wohlin. I appreciate your valuable time and comments on the thesis. My paranymphs Moritz van Heesch and Timo Meinen deserve special thanks for assisting me during the defense ceremony.

I also appreciate the fruitful discussions and the good advice from my friends Binne and Widura Schwittek, and Romina and Claas Rettinghausen. You guys are great consultants. I thank Heinz and Heinz Bömler for letting me and my family live at such a wonderful place, which has been a rich source of inspiration for me.

I thank all the other people who cooperated with me in the last years: Veli-Peka Eloranta, Kai Koskimies, Rich Hilliard, and Antony Tang. You have certainly contributed to the success of my research projects.

Finally, I want to thank the students who put so much effort into implementing and improving the tool support for my research approaches. In particular Ben Ripkens, Christian Manteuffel, Martin Verspai, Stefan Arians, and Michel de Jong.

I believe that the events of life are determined by the people we meet and accompany for a while. I am full of appreciation for all the great people I have met. You all are indispensable for where I am and who I am now.

Uwe van Heesch
Groningen
November 12, 2012

Part I

Foundations

Chapter 1

Introduction

The front cover of this dissertation shows a snapshot of an approximately 100-year old building[1], taken during construction work. I chose this picture to explain some of the main problems with software architecture decisions, using an analogy to building architecture decisions.

During the construction work shown in the picture, an intermediate floor level was built and parts of an outer wall were replaced by floor-to-ceiling windows, to convert parts of the building into living space. In its original form, the right part of the building did not have any windows or intermediate floors, and multiple iron rods went through the inside, attached to the outer walls using plate-sized metal washers and nuts. The outer walls are up to one meter thick and get slightly thinner from the bottom to the top of the building. The decisions to design the building this way were made by an anonymous architect in the late 19th century.

The building used to be part of a watermill. It served as a silo, which was used to store tons of linseed that were processed into oil. The back cover of this dissertation shows a picture taken in that time. Without this context information, the thickness of its walls, the iron rods, and the fact that no windows and intermediate levels were present, can hardly be understood. The river that powered the mill was redirected in 1933; today, there is almost no sign that the building was ever part of a watermill.

Nevertheless, when planning the construction work shown on the front cover, the architect was confronted with the decisions of his or her predecessor. Questions like the following came up: *why are the walls so abnormally thick?*; *what is the purpose of the iron rods that go through the building?*; *what are the consequences if the iron rods are removed?* In short: *what is the rationale behind the original architect's decisions?* These questions had to be answered before changing the architecture of the building. In this case, the thickness of the walls can be explained by the tons of seed that pushed against the walls from inside the silo; the iron rods and washers attached to the walls served as additional protection against the pressure of the seeds, by holding two opposing sides of the silo together. Thus, the changes shown on the cover picture could be made without compromising the structural stability of the building, because the seed does not press against the walls anymore. At the same time, the architectural impact of the changes was so huge that the building could not be used for its original purpose again. The original architectural integrity is destroyed.

Similar questions and problems can arise during the evolution of software systems.

[1] The building is part of the "Viller Mühle" complex in Goch, Germany.

They are even more likely to arise, because software systems are substantially more complex than the building shown on the cover. While the architecture decisions made for the building can be understood from the fact that it was designed to be used as a silo, software systems consist of so many intertwined architectural elements that the rationale behind each and every one cannot be understood just by knowing the system's purpose. This is particularly problematic, because software is subject to continuous change during its lifetime, for instance to maintain the system, or to adapt it to changing requirements. A software architect, who is designing the change, might ask questions like: *what is the purpose of this component?*; *why was this subsystem designed like this, instead of doing it a different, apparently more efficient way?*; *what are the dependencies between two subsystems?* Contemporary software architecture documentation rarely answers such questions regarding the purpose, or the rationale of architectural elements.

The title of this dissertation is an allusion to Bosch's position paper "Software architecture: The next step" (Bosch 2004), in which he promotes the explicit documentation of software architecture decisions as the next step in software architecture research. The fundamental principle of this thesis is that the documentation of architecture decisions remedies a great part of the earlier hinted problem, which Bosch refers to as architectural knowledge vaporization (Bosch 2004). Bosch, and subsequently other researchers in the software architecture field, have introduced the concept of architecture decisions as first-class entities in architecture description (Bosch 2004, Tyree and Akerman 2005, Kruchten 2004a), and proposed approaches for managing and maintaining decisions (Jansen 2008, Zimmermann, Gschwind, Küster, Leymann and Schuster 2007, Farenhorst and de Boer 2009). As a next step in architecture decision research, we need to understand better how architecture decisions are made in practice, and how they can be modeled to integrate with other types of architectural description. Additionally, modeling architecture decisions can have immediate benefits for the initial architectural design process and it can support architecture evaluation, as I will show in the remainder of this dissertation.

This dissertation deals with architecture decisions: how they are made, how they can be modeled to support architecture decision-making, and how they can be reviewed efficiently. It is a next step towards tapping the full potential of architecture decisions in the software architecture research field.

In the remainder of this chapter, I give a brief introduction to software architecture and software architecture decisions.

1.1 Software architecture

Every software system has an architecture (Bass et al. 2003, Rozanski and Woods 2005, Taylor et al. 2009). Yet, no agreement has been reached on how to exactly define software architecture. The Software Engineering Institute has collected far more than 100 definitions for software architecture from the software architecture community[2]. Rather

[2]see `http://www.sei.cmu.edu/architecture/start/glossary/community.cfm`

than adding yet another definition to this long list, I briefly discuss two of the most adopted definitions in the field.

The first definition takes a purely product-oriented perspective on architecture: "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." (Bass et al. 2003). Software elements can be subsystems, layers, packages, or components in the sense of component-based software engineering, for instance. The properties of these elements, as referred to in the definition, are behavioral properties (what the system does), and quality properties (how the system does it) (Rozanski and Woods 2005). Finally, the definition takes into account the relationships among the software elements, e.g. how they interact, or depend on each other.

The second architecture definition is taken from the international standard for architecture description, ISO/IEC/IEEE 42010, which defines architecture as the "fundamental concepts or properties of a system in its environment embodied in its elements, relationships, and in the principles of its design and evolution" (ISO/IEC/IEEE 2011). This definition is wider than the previous one. Apart from elements, relationships, and properties, it takes the principles into account that govern the design and evolution of a system. A principle is not necessarily of technical nature. An example for a non-technical principle could be: *maximize financial benefit to the company*. In contrast to the previous definition, which considers only the end product (the software system) in the architecture definition, ISO/IEC/IEEE 42010 acknowledges that the main propositions that serve as the logical fundament for all decisions made during the design and evolution of the system, are also part of the system's architecture.

In this dissertation, the second definition of software architecture is used. The next section elaborates the concept of architecture decisions.

1.2 Architecture decisions

Long before ISO/IEC/IEEE 42010 was formulated, Perry and Wolf described the rationale that went into the various choices made in designing an architecture, as an integral part of the architecture (Perry and Wolf 1992). Although this view was adopted by researchers and practitioners (e.g. (Kruchten 1995, Garlan et al. 1997, IEEE 2000)), there was little advice on how to preserve the rationale that went into an architecture. The focus of most architecture approaches was on documenting architectural design elements, often manifested in architectural views.

Starting in 2004, the rationale that went into the architectural choices came into the focus of software architecture research. In his previously mentioned position paper "Software Architecture: The Next Step", closely followed by Tyree, Akerman, and Kruchten (Tyree and Akerman 2005, Kruchten 2004a), Bosch suggested to treat architecture decisions as first-class entities in software architecture representation (Bosch 2004).

Architecture decision representations were primarily introduced to avoid the loss of

information about decisions and the decision-making process, e.g. patterns or architectural styles that were applied, problems that had to be solved, considered alternatives, addressed requirements, and which implications a decision has. This information has been considered vital to ease the communication between stakeholders (Tyree and Akerman 2005), and also particularly during system evolution, to make sure that earlier decisions are not violated (Bosch 2004). Since then, the management of architectural knowledge, including architecture decisions as a major part, has been investigated by many researchers in the software architecture field, e.g. (Jansen 2008, Farenhorst and de Boer 2009, Clerc 2011). Since 2006, the annual workshop on SHAring and Reusing architectural Knowledge (SHARK Workshop 2012) takes place to advance architecture knowledge management approaches in practice and academia.

This dissertation follows up on the idea of treating architecture decisions as first-class entities in architecture description. After analyzing how architecture decisions are made in practice, it illustrates a way to practically realize decision description as part of a software architecture. Finally, it shows that decision modeling can support the architectural design process and architecture evaluation.

Chapter 2

Research design

This chapter presents the main problem addressed in the dissertation, the research methodology used, and the research questions dealt with. Additionally, the usage of empirical research methods in this research project is discussed.

2.1 Problem Statement

Other researchers have made great progress in describing architecture decisions (Tyree and Akerman 2005, Kruchten 2004a, Jansen 2008) and different approaches and tools were proposed to model and manage architecture decisions, e.g. by Jansen et al. (Jansen, de Vries, Avgeriou and van Veelen 2008), Capilla et al. (Capilla et al. 2007), Tang et al. (Tang et al. 2007), Zimmermann et al. (Zimmermann et al. 2008), and Fahrenhorst and van Vliet (Farenhorst and van Vliet 2009). All these approaches and tools satisfy some stakeholder concerns in architecture decision modeling and documentation (Appendix B.1 presents a comprehensive list of decision-related concerns), but none of them satisfies all. As an example, the decision documentation template proposed by Tyree and Akerman (Tyree and Akerman 2005) is a good way of documenting the rationale behind a single architecture decision, but it is ineffective for getting an overview over all decisions made, or for performing impact analyses. On top of that, with few exceptions, such as the decision view proposed by Kruchten, Capilla, and Dueñas (Kruchten et al. 2009), most approaches to architecture decision documentation do not integrate well with other types of architecture descriptions (Tang, Avgeriou, Jansen, Capilla and Ali Babar 2010), which are typically organized using different architectural viewpoints.

Apart from preserving information about the architecture decisions made, e.g. for later analysis during architecture evolution, architecture decision modeling should support architectural design activities. By *architectural design activities*, I refer to architectural analysis, architectural synthesis, and architectural evaluation, as defined in Hofmeister et al.'s general model of architecture design (Hofmeister et al. 2007).

The following statement summarizes the previously mentioned problems. It is the basis for the research questions described in Chapter 2.3:

"Existing architecture decision modeling approaches do not satisfy all stakeholders' concerns, they do not integrate with viewpoint-based architecture descriptions, and they do not optimally support architectural analysis, architectural synthesis, and architectural evaluation."

2.2 Design science as research methodology

The research project, documented in this dissertation, adopts the design science framework described by Wieringa (Wieringa 2009). Design science is a technology-oriented discipline that seeks to create or improve "things" that serve human purposes (March and Smith 1995, Wieringa 2009). In the original definition, design science comprises two central activities: *building* and *evaluating* for the purpose of contributing to a domain's knowledge base. Building refers to the construction of an artifact for a specific human purpose; evaluation determines how well the artifact suits this purpose (March and Smith 1995). Wieringa's framework contains the additional activity *problem investigation*; it seeks to understand the given problem without changing it yet (Wieringa 2009).

Although design science originated from the information systems field (March and Smith 1995), it is not limited to this domain. In fact, the design science cycle, which is comprised of the three activities problem investigation, building, and evaluation, is very similar to the software architecture design process. According to Falessi et al., software architecture design includes the activities *understand the problem*, *find a solution for the problem*, and *evaluate the solution* (Falessi et al. 2010). Hofmeister et al. refer to the same activities as *architectural analysis*, *architectural synthesis*, and *architectural evaluation* (Hofmeister et al. 2007).

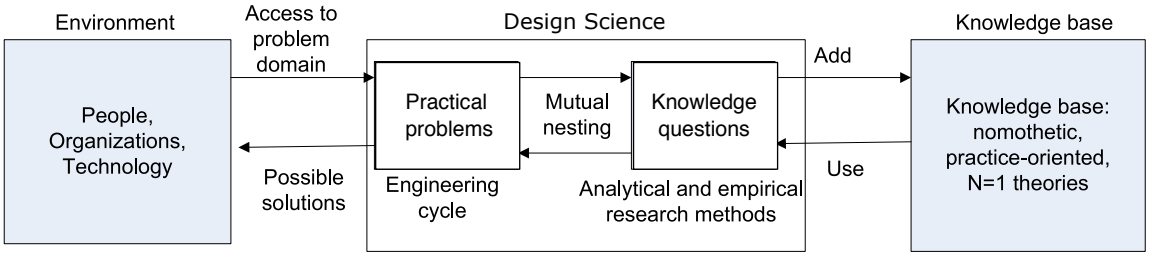


Figure 2.1: *Design science framework adopted from Wieringa (Wieringa 2009)*

Figure 2.1 conceptualizes the design science framework developed by Wieringa, as a refinement of Hevner et al.'s framework (Hevner et al. 2004). Design science is inherently practice-oriented, taking needs from the environment as a starting point. Wieringa distinguishes two types of topics, design science deals with: practical problems, and knowledge questions. A practical problem is defined as "a difference between the way the world is experienced by stakeholders and the way they would like it (the world) to be"; a knowledge question is "a difference between current knowledge of stakeholders about the world and what they would like to know" (Wieringa 2009).

An example of a practical problem in the software architecture domain could be: *design an architecture description language (ADL) for distributed machine-control systems*. It is a practical problem, because its aim is to change the world by creating something new; however, it also implicitly entails at least two knowledge questions: *what are the stakeholders' concerns in the ADL to be designed?*; and for evaluation purposes: *does the designed ADL satisfy the stakeholders' concerns?* These are knowledge questions, because they do

not aim at changing the world, but at changing the knowledge about the world. This is one example for the nested nature of practical problems and knowledge questions, as also expressed in Figure 2.1.

Design science projects seek to solve existing practical problems from the environment. In order to achieve that, they either apply knowledge from a domain's knowledge base to answer knowledge questions; or, by conducting original research, they contribute to the knowledge base of a domain.

Design science is iterative: The researcher analyzes a practical problem from the environment, proposes a solution, evaluates the solution, and then starts over again. The analysis of a practical problem, and the evaluation of the solution are both knowledge questions. Design science researchers refer to these iterative activities as design cycle (Hevner 2007). The repetitive rounds through the cycle stem from the fact that the evaluation activity may uncover that aspects of the original problem were not addressed, or additional practical problems or knowledge questions could emerge.

The design science framework is particularly suitable for describing long-term research like PhD projects, because it allows to present the evolution of research questions and solutions at the same time. While a PhD project has an initial problem statement as a starting point, more concrete research questions usually emerge when the researcher gains a deeper understanding of the problem and develops partial solutions, which in turns lead to new research questions. In the following section, this PhD project is described using Wieringa's design science framework (Wieringa 2009).

2.3 Practical problems and knowledge questions

This section explains the practical problems and knowledge questions addressed in this PhD project, and how they emerged from each other. Figure 2.2 visualizes the problems and questions; grey boxes represent knowledge questions, white boxes represent practical problems. Furthermore, hollow arrows denote sequence, solid arrows denote decomposition. In the remainder of this section, I refer to both practical problems and knowledge questions as research questions. The research questions are labeled with numbers from one to five. With the exception of number four, each research question is decomposed into two to four sub-questions, labelled with letters from a to d.

The problem statement in Section 2.1 describes that contemporary decision modeling approaches do not satisfy all stakeholder's concerns, do not integrate in viewpoint-based architecture descriptions, and that they do not support architecture activities. Before we looked into ways on how to improve decision modeling approaches in order to address these problems, we first had to understand how decisions are really made. Therefore, RQ 1 (*How are ADs made?*) is concerned with finding out how architecture decisions are made in practice, i.e. how architects reason and which deficiencies exist in the reasoning process of inexperienced designers. RQ 1 is decomposed into two sub-questions. In order to explore the innate decision-making process, which is unbiased by architecting approaches and company policies, in RQ 1.a (*How do students make ADs?*),

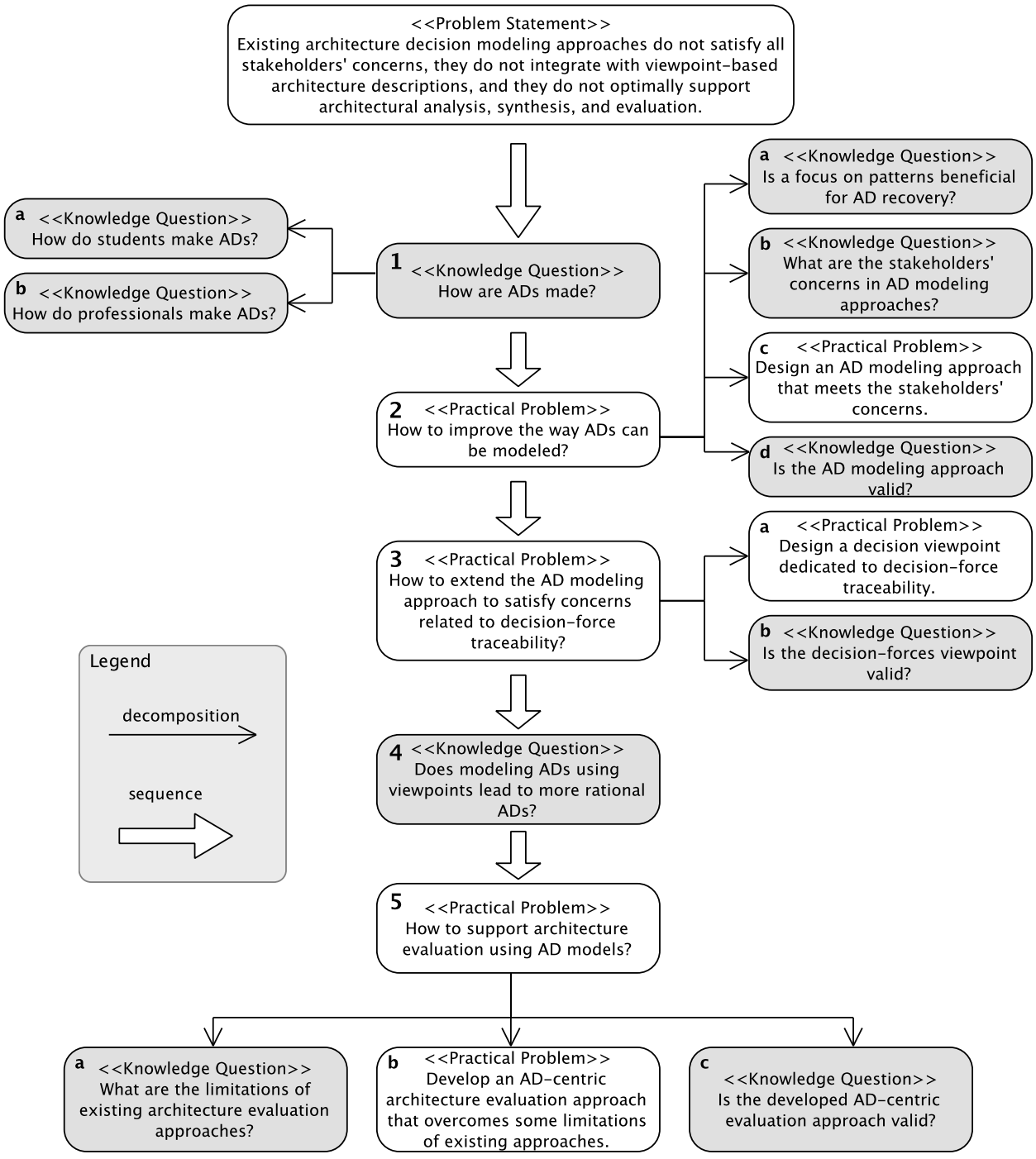


Figure 2.2: *Research questions addressed in this research project*

we studied the most inexperienced subjects from the target population: final-year software engineering students. Studying beginners helped us to identify problems and shortcomings in the decision-making process. RQ 1.b (*How do professionals make ADs?*) explored the professional decision-making process by investigating how architects from the industry make decisions.

After gaining a good understanding of the decision-making process, including typical shortcomings, we started addressing the problem of how architecture decision mod-

eling can be improved in a way that it integrates in viewpoint-based architecture description, while satisfying all stakeholders' concerns in decision description. Hence, RQ 2 is *How to improve the way ADs can be modeled?* Following Wieringa's classification (Wieringa 2009), RQ 2 is a practical problem. It requires a change in the world, rather than a change in knowledge. RQ 2 is decomposed into four sub-questions RQ 2.a - RQ 2.d.

As a first attempt to make architecture decision modeling more efficient, we thought about a method to capture decisions and the rationale behind them in a way that requires less effort by the architects than filling in decision templates during the architecting process. Inspired by Harrison et al. (Harrison et al. 2007), who describe the benefits of using software patterns to capture architecture decisions, we dealt with the question if decisions can be effectively recovered by focussing on identifying applied patterns in an architecture. Many software systems are designed using patterns, which provide rich information about the applied solution and the rationale behind the solution in the form of a problem description and the forces that influence the selection of a solution. If an applied pattern can be identified in an architectural design, then a great part of the rationale that went into the decision can be deduced from the pattern description. Therefore, we hypothesized that a pattern focus during architecture decision recovery significantly increases the quality and the quantity of recovered architecture decisions, compared to decision recovery that has no special focus (RQ 2.a).

Although patterns can be used to recover and describe architecture decisions effectively, many other concerns in decision modeling remained unaddressed. As a consequence, we decided to develop a new architecture decision modeling approach to solve the central research problem stated in Section 2.1. RQ 2.b - RQ 2.d refer to the three design science activities problem investigation, building, and evaluation. RQ 2.b (*What are the stakeholder concerns in decision modeling approaches?*) is the problem investigation; RQ 2.c (*Design an AD modeling approach that meets the SH concerns*) refers to the building activity; and RQ 2.d (*Is the AD modeling approach valid?*) refers to the evaluation activity. As a result of RQ 2, we created a framework for architecture decisions, which can be used to model ADs using multiple viewpoints, each of which addressing specific stakeholder concerns.

The evaluation of the framework in RQ 2.d (*Is the AD modeling approach valid?*) provided evidence that the viewpoints satisfy most of the concerns identified as a result of RQ 2.b (*What are the stakeholder concerns in decision modeling approaches?*), but at the same time it became evident that concerns related to decision-requirements traceability and concerns related to decision-design traceability could not be addressed by the framework. Decision-requirements traceability is the subject of RQ 3, which is a practical problem concerned with the extension of the decision modeling framework to satisfy concerns related to decision-requirements traceability; decision-design traceability is subject to our ongoing research described in Section 10.2.

In the studies conducted to find out how ADs are made in practice (RQ 1), we had found out that architecture decisions are not only driven by requirements, but also by additional influencing factors, e.g. the development team's previous experience with

specific architectural solutions. To acknowledge this finding, we extended RQ 3 by addressing decision-force traceability rather than only addressing decision-requirements traceability. Decision forces capture any non-trivial impact on an architect when making decisions. Thus, even though all architecture-significant requirements are decisions forces, there are numerous other kinds of forces. Decision forces are discussed in detail in Chapter 7. According to the design science cycle, RQ3 is decomposed into a building activity (RQ 3.a: *Design a decision viewpoint dedicated to decision-force traceability*), and an evaluation activity (RQ 3.b: *Is the decision-forces viewpoint valid?*). In this case, the problem investigation had already taken place as part of RQ 2.d (*Is the AD modeling approach valid?*).

Apart from decision-design traceability, the developed framework for architecture decisions, and the additional viewpoint for decision-force traceability, satisfy all previously identified stakeholder concerns in decision modeling. In RQ 4 and RQ 5, we examined if decision viewpoints can also support architectural analysis, synthesis, and evaluation, as described in the problem statement in Section 2.1. In RQ 4, we investigated if architecture decision modeling, using our decision framework and the forces-viewpoint extension, can support more rational decision-making regarding architectural analysis and architectural synthesis. The evaluation of the forces viewpoint in RQ 3.b (*Is the decision-forces viewpoint valid?*) had already given us first indications for the supportive effect of decision view modeling on the decision-making process.

The results from RQ 4 showed that the decision framework and the forces viewpoint provided at least partial support for architectural analysis and architectural synthesis. Architecture evaluation is addressed in RQ 5, which is a practical problem concerned with the question how architecture evaluation can be supported by means of decision models. We started from the premise that recovering decision views after-the-fact can support evaluating how well decisions address the relevant decision forces. The validation of the decision framework (RQ 2.d: *Is the AD modeling approach valid?*) had shown us already that particularly the decision relationship viewpoint provides support for architecture reviews, as it was found to be well suited for identifying important and critical decisions, performing impact analyses, and finding dependencies between decisions. As a result, in order to answer RQ 5, we used the decision relationship viewpoint and the decision detail viewpoint for performing systematic architecture evaluation using ADs as primary targets of the evaluation. The concept of forces, which was developed as part of RQ 3 (*How to extend the AD modeling approach to satisfy concerns related to decision-force traceability?*), and which played an important role in the results of RQ 4 (*Does modeling ADs using viewpoints lead to more rational ADs?*), was adopted as a central concept for capturing and evaluating the rationale that went into the architecture decisions under examination. The answer to RQ 5 is an architecture evaluation approach that overcomes some of the limitations of existing evaluation approaches, by using decision viewpoints and forces as central concepts.

This section outlined the thread of research questions addressed in this dissertation. The answers to the research questions contribute to the software architecture knowledge base (see Figure 2.1); furthermore, the answers to each question were used as input for

the subsequent research questions, as described in this section.

2.4 Using empiricism to answer knowledge questions

The previous section summarized the research questions addressed in this dissertation. As the overview of research questions in Figure 2.2 shows, the majority of research (sub-) questions, we dealt with, fall into the category of knowledge questions. In contrast to practical problems, which cause a change in the world, knowledge problems seek to provide knowledge relevant to a particular subject domain (Wieringa 2009). In this section, I discuss the usage of empiricism to answer the stated knowledge questions.

Compared to its use in other disciplines, like medicine or social sciences, the use of empiricism is rather novel in the software engineering and software architecture fields. Nevertheless, it has been suggested as a means to evaluate and understand methods, processes, techniques, and tools we develop and use, by many researchers in the field, e.g. by (Perry et al. 2000, Wohlin et al. 2012, Kitchenham et al. 2002).

In spite of the fact that the software engineering discipline is lacking commonly accepted guidance on how to rigorously conduct different types of scientific studies (Shaw 2002), a number of guidelines have been published, particularly for empirical research in software engineering (Wohlin et al. 2012, Wohlin et al. 2003, Höst and Runeson 2007, Easterbrook et al. 2008). In the following, we describe the empirical methods covered by these guidelines, and the criteria that should be taken into consideration when selecting between them.

Experiment: Experiments are particularly suitable for establishing cause-effect relationships between multiple study variables (Wohlin et al. 2003). They require the specification of one or more hypotheses, tested in a controlled environment, in which confounding factors can be eliminated to the best possible degree (Easterbrook et al. 2008, Wohlin et al. 2012). Experiments are usually conducted to confirm the researcher's view on contemporary events (Yin 2003).

Survey: Surveys are suitable for investigating broad subject populations (Easterbrook et al. 2008). They are usually conducted using individual interviews, group interviews, or questionnaires as data collection methods. Depending on the concrete research design, surveys can deliver qualitative or quantitative data, be descriptive, explanatory, or explorative (Wohlin et al. 2012, Wohlin et al. 2003). In contrast to experiments, surveys do not require control over behavioral events (Yin 2003), but they bare the risk of sampling bias, if the researcher selects subjects that are likely to confirm his or her research conjecture, rather than striving for a sample that is representative for the target population. The thorough design of questions is important in surveys. Badly designed or indicative questions are threats to the internal validity of the survey results (Easterbrook et al. 2008).

Case study: Case studies are suitable for investigating contemporary phenomena in their real-life context. They enable the researcher to gain a deep understanding of

the case under study (Easterbrook et al. 2008). Case studies are mostly explorative in nature, answering how and why questions (Yin 2003, Easterbrook et al. 2008). Like surveys, they do not require control over behavioral events. Case studies are easier to plan than experiments or surveys, but they are also more difficult to generalize, as the subject population is usually small and confounding factors can hardly be eliminated (Wohlin et al. 2012, Wohlin et al. 2003).

Kitchenham et al. describe a variant of case study research, we refer to as comparative case studies, used for method and tool evaluation (Kitchenham et al. 1995). They combine experiments, which are usually the first choice for confirmative research questions, with the advantages of case studies, which allow to gain a more holistic view of a phenomenon in its natural environment (rather than a laboratory environment). Yet, as Kitchenham et al. point out, single case studies are inappropriate for doing comparisons, because they are lacking a reference that can be used as a basis for the comparison (Kitchenham et al. 1995). Therefore, Kitchenham et al. propose a study design that relies on multiple case studies, in which the results of a subset of the cases serve as a baseline for the comparison, while the tool or method under evaluation is only applied in the rest of the cases.

Grounded theory: Grounded theory is an empirical research method that generates theories, mainly from qualitative data (Glaser and Strauss 1967). When applying grounded theory, and its associated constant comparative method (Glaser 1965), the researchers develop theories about phenomena exclusively based on the collected data, following a rigorous analysis process, in which temporary theories that emerged from pieces of collected data are constantly checked against the rest of the collected data. Thus, grounded theory is inherently explorative. Using grounded theory, the researchers are not looking for evidence supporting previously formulated theories, but they seek to produce new theories. Used as a data analysis method, grounded theory can be combined with the previously mentioned research methods, as long as the collected data holistically captures the phenomenon under study and was not pre-filtered, e.g. with respect to specific research questions.

Compared to experiments, surveys and case studies, grounded theory is rarely used in software engineering research. Nevertheless, recent studies have demonstrated the applicability of grounded theory in this field (Urquhart et al. 2010, Adolph et al. 2011), as well.

Each of the empirical methods, described above, was used to answer specific knowledge questions posed in this research project. With the exception of grounded theory, the methods listed above are currently the most prevalent empirical methods used in software engineering research. Other methods like action research (Avison et al. 1999, Easterbrook et al. 2008) and ethnography (Sharp et al. 2010) have also been suggested for use in software engineering. Yet, they did not serve the purposes of the knowledge questions we identified.

Table 2.1: *Empirical methods used to answer the knowledge questions*

Code	Knowledge question	Empirical method	Described in
RQ 1.a	How do students make ADs?	Survey	Section 3.3.2
RQ 1.b	How do professionals make ADs?	Survey	Section 4.3
RQ 2.a	Is a focus on patterns beneficial for AD recovery?	Controlled experiment	Section 5.3
RQ 2.b	What are the SH concerns in AD modeling approaches?	Literature review	Appendix B.1
RQ 2.d	Is the AD modeling approach valid?	Case study	Section 6.4
RQ 3.b	Is the decision-forces viewpoint valid?	Multiple-case study, grounded theory	Section 7.4
RQ 4	Does modeling ADs using viewpoints lead to more rational ADs?	Comparative case studies, grounded theory	Section 8.2
RQ 5.a	What are the limitations of existing architecture evaluation approaches?	Literature review	Section 9.1
RQ 5.c	Is the developed AD-centric evaluation approach valid?	Survey	Section 9.4

Table 2.1 shows which research methods were used for the knowledge questions described in Figure 2.2. Additionally, it contains references to the sections in this dissertation, where the study design using the respective method is described. It must be noted that the literature reviews conducted to answer RQ 2.b and RQ 5.a were not systematic, but covered only sources we regarded as particularly important.

2.5 Overview of this dissertation

This dissertation is divided into multiple parts. The first part, *Foundations*, includes the introduction to software architecture and architecture decisions, as well as the research design, which were presented earlier in this chapter. The main body of the dissertation contains four additional parts: *Understanding ADs*, *Modeling ADs*, *Supporting ADs*, and *Evaluating ADs*. Table 2.2 shows the research questions and the chapters, in which they are addressed.

Chapters three to nine are based on scientific journal or conference articles, which are either published, or currently under revision. In the following, each chapter is briefly outlined.

Chapter 3 is based on a long, peer-reviewed conference paper in the proceed-

Table 2.2: *Overview*

Research question	**Chapter**
Part 1: Understanding architecture decisions	
RQ1: How are ADs made?	Chapter 3
	Chapter 4
Part 2: Modeling architecture decisions	
RQ 2: How to improve the way ADs can be modeled?	Chapter 5
	Chapter 6
RQ 3: How to extend the AD modeling approach to satisfy concerns related to decision-force traceability?	Chapter 7
Part 3: Supporting architecture decisions	
RQ 4: Does modeling ADs using viewpoints lead to more rational ADs?	Chapter 8
Part 4: Evaluating architecture decisions	
RQ 5: How to support architecture evaluation using AD models?	Chapter 9

ings of the fourth European Conference on Software Architecture (van Heesch and Avgeriou 2010). The chapter reports on a study conducted with students to understand their innate reasoning process during architectural design. *Chapter 4* is based on a long, peer-reviewed conference paper in the proceedings of the ninth Working IEEE/IFIP Conference on Software Architecture (van Heesch and Avgeriou 2011). The chapter describes a survey, conducted with industrial software architects, to find out how they reason when making architecture decisions in real software projects.

Chapter 5, based on a peer-reviewed journal publication in Science of Computer Programming (van Heesch, Avgeriou, Zdun and Harrison 2012), is a joint work with Uwe Zdun and Neil Harrison. The chapter describes a controlled experiment conducted to find out if a focus on identifying applied patterns during architecture decision recovery leads to higher quality and quantity of recovered ADs. I was the lead author in this publication and designed the entire study upfront. The co-authors assisted during the execution of the study, contributed with ideas, and reviewed the manuscript.

Chapters 6 and 7 are a joint work with Rich Hilliard. *Chapter 6* presents a documentation framework for architecture decisions using the conventions of ISO/IEC/IEEE 42010 (ISO/IEC/IEEE 2011). It is based on a peer-reviewed publication in the Journal of Systems and Software (van Heesch, Avgeriou and Hilliard 2012a). *Chapter 7* presents an extension to this work, an architectural viewpoint dedicated to architecture decision - forces traceability. It is based on a peer-reviewed long conference paper in the Joint 10th Working IEEE/IFIP Conference on Software Architecture & 6th European Conference on Software Architecture (van Heesch, Avgeriou and Hilliard 2012b). Apart from being the lead author of both publications, I developed the entire decision viewpoint framework and designed the empirical studies conducted. The co-authors of the publications contributed with ideas and reviewed the manuscripts.

Chapter 8, which is in revision for the Journal of Systems and Software, presents a comparative multiple-case study, conducted to find out if modeling ADs, using architecture decision viewpoints, supports senior software engineering students in following a rational design process. The chapter is a joint work with Antony Tang. Apart from being the lead author, I designed the entire study. The co-authors contributed with ideas and assisted during the data analysis.

Chapter 9 describes a decision-centric architecture review method, which was developed together with Veli-Pekka Eloranta, Kai Koskimies, and Neil Harrison. The chapter is currently in revision for the IEEE Software Magazine. Veli-Pekka Eloranta and myself are the lead authors of this publication. Apart from this, all authors contributed equally to the development of the method.

The dissertation ends with Part 5, *Reflection and Conclusions*. It summarizes the answers to the research questions raised in Section 2.3 and outlines areas of ongoing and future work.

Part II

Understanding architecture decisions

Based on: U. van Heesch and P. Avgeriou – *"Naive architecting-understanding the reasoning process of students: a descriptive survey"*, Proceedings of the 4th European conference on Software architecture, pp. 24-37, 2010.

Chapter 3

Naive architecting - understanding the decision-making process of students

Abstract

Software architecting entails making architecture decisions, which requires a lot of experience and expertise. Current literature contains several methods and processes to support architects with architecture design, documentation, and evaluation, but not with the design reasoning involved in decision-making. In order to derive a systematic reasoning process, we need to understand the current state of practice and propose ways to improve it. In this chapter, we present the results of a survey that was conducted with undergraduate software engineering students, aiming to understand the innate reasoning process during architecting. The results of the survey are compared to the architecture literature, in order to identify promising directions towards systematic reasoning processes.

3.1 Motivation

One of the responsibilities of software architects is to make decisions, which are usually called architecture decisions (Bosch 2004, Jansen and Bosch 2005, van der Ven, Jansen, Nijhuis and Bosch 2006) and determine the overall structure and behavior of the system. Making architecture decisions involves understanding and addressing relevant requirements, business goals and issues, identifying and choosing among alternative solutions, while adhering to constraints and mitigating risks. Architecture decisions form the basis for all other detailed decisions and are crucial for the success or failure of the whole project. This decision-making process is one of the major challenges during architecting, since it requires a lot of experience and expertise by the architect.

Various methods exist to support software architects in their work. Hofmeister et al. derived a common model for architecture design from five industrial approaches (Hofmeister et al. 2007), including the Rational Unified Process (Kruchten 2004b) and Attribute-Driven Design (Bass et al. 2003). Other approaches deal with documenting the architecture in terms of multiple architectural views or with the help of architecture frameworks (IEEE 2000, Clements et al. 2010, Kruchten 1995). Furthermore, different methods exist to support the systematic evaluation of architectures (Kazman et al. 2000, Kazman et al. 1994, Williams and Smith 2002). More recently, some approaches proposed the documentation of the actual decisions as first-class entities by defining their attributes and relations (Bosch 2004, Tyree and Akerman 2005). However, all of these approaches deal with the core part of architecting: prioritizing ar-

chitecturally significant requirements, selecting architecture patterns, styles and tactics, partitioning the system into components and connectors, assessing the design and documenting the result with architectural views, frameworks and architecture description languages. In contrast, there has been very little research on the reasoning part of the decision-making process; one can only find fragments about sound reasoning in the literature.

Recent work emphasizes the importance of design reasoning and design rationale (Bosch 2004, Tang et al. 2006). Ideally, a systematic reasoning process can shorten the gap between experienced and inexperienced architects: design reasoning can support designers step-by-step in making sound decisions and subsequently documenting the rationale behind them as first class entities. However, so far architects are not trained on how to reason: making architecture decisions is often described as an ad-hoc creative process (Bosch and Molin 1999, Zdun 2007, Zimmermann et al. 2008) that relies heavily on the personal experience and expertise of the architect. Research is required to explore the current state of practice in design reasoning and subsequently to find ways to enhance it.

Our work is towards this direction: investigating how the reasoning process takes place and identifying potential areas for improvement. This can be done either by studying beginners (bottom-up), or experienced architects (top-down). The former case allows to establish the baseline reasoning process that is based on common sense instead of experience. The latter case allows to discover best practices in successful architecting examples and to synthesize them into an ideal reasoning process. Eventually, one can propose an approach to close the gap between the baseline and the ideal process and package it appropriately to train current or future architects.

This chapter deals with the former case; the latter case is presented in Chapter 4. In particular, we have studied the most inexperienced subjects: software engineering students. We asked 22 students to design an architecture for a large web application. After that, the students were interviewed about the way they thought and acted to come up with a software architecture. As a result, we identified the basic reasoning process of inexperienced designers, which we compared to established architecting processes in the literature, in order to come up with promising directions for improvement.

The rest of this chapter is organized as follows. Section 3.2 presents related work. In Section 3.3, the design of the study is introduced. The next section presents an analysis of the results, which are interpreted in Section 3.5. The chapter ends with conclusions and directions for further work.

3.2 Related Work

The survey presented in this chapter is related to the software architecture research field, namely architecting processes, architecting practice in the industry, and design reasoning.

Hofmeister et al. derive a general model of architecture design from five industrial

approaches (Hofmeister et al. 2007). They identify the following common activities: *architectural analysis* is concerned with identifying architecturally significant requirements from architectural concerns and system contexts; *architectural synthesis* is the activity of finding candidate solutions for architecturally significant requirements; architectural evaluation makes sure that the candidate solutions are the right ones.

Jansen et al. specialize this generic model from the perspective of architecture decisions (Jansen, Bosch and Avgeriou 2008). They describe the architecting process as a cycle of activities that are followed iteratively until the architecture is complete. In accordance with Hofmeister et al.'s categorization, in architectural analysis, the problem space is scoped down to problems that can be solved by single architecture decisions. Candidate decisions are proposed during architectural synthesis, while decisions are chosen during architectural evaluations, which also entails modifying and describing the architecture in multiple architectural views. In addition to Hofmeister et al.'s approach, which focuses mainly on architecting activities and artifacts, Jansen et al. indicate reasoning processes within the activities.

Various studies have attempted to define the role of software architects in the industry (Kruchten 1999, Clerc et al. 2007, Hoorn et al. 2011, Kruchten 2008, Clements et al. 2007). Clerc et al. have conducted survey-based research (Clerc et al. 2007) to gain insights in the daily working processes of architecture practitioners. They found out that architecture use cases (van der Ven, Jansen, Avgeriou and Hammer 2006) concerning risk assessment and requirements trade-off analysis are not regarded as particularly important by the architects. In contrast, use cases concerned with requirements, architecture design and implementation, and the traceability among these were rated as important. The authors reckon that the architects' workflow follows a linear approach to designing architecture that satisfies the requirements subsequently.

In a different survey, Hoorn et al. (Hoorn et al. 2011) describe that more experienced architects, in terms of working years, are more often involved in auditing activities and quality assurance. Kruchten defines the typical roles and responsibilities that architects should take in software projects (Kruchten 2008). Besides making architecture decisions, other central activities of architects include maintaining the architectural integrity, risk assessment and risk mitigation.

Finally, Clements et al. compare duties, skills, and knowledge of software architects from the perspectives of literature, education and practice (Clements et al. 2007). They found that architecture evaluation and analysis are regarded as less important in architecture practice, whereas knowledge of technologies and platforms, as well as technology-related duties are regarded more important in architecture practice than in the literature and education. We will revisit these results on architecting practice and relate them to our findings in Section 3.6.

The significance of design reasoning in software architecture has been recently emphasized. Tang and Lago describe design reasoning tactics (Tang and Lago 2010) to support architects in structuring architectural problems and extracting design issues. In his previous work, Tang declares the importance of design reasoning and design rationale in the area of software architecture (Tang et al. 2006, Tang et al. 2008). It supports archi-

tects in making well-founded decisions and provides guidance to explore and manage the solution space. They state that the use of a reasoning approach significantly improves the quality of architectural design, especially for inexperienced architects (Tang et al. 2006).

3.3 Design of the Study

3.3.1 Goal

The goal of the study is to get insight into the innate reasoning that students follow while they are architecting. To make this goal more concrete, we need to consider the fundamental reasoning activities that take place during the architecting process. As a reference architecting process, we use the one defined by Jansen et al. (Jansen, Bosch and Avgeriou 2008), which explicitly takes into account the reasoning aspects and maps onto the process of Hofmeister et al. (see Section 3.2). We thus refine our research goal into the following three research questions:

RQ1: How do students scope and prioritize the problem space during architectural analysis?

RQ2: How do students propose solutions during architectural synthesis?

RQ3: How do students choose among solutions during architectural evaluation?

RQ1 is concerned with finding out how students scope and prioritize requirements and issues to define concrete problems that are small enough to be addressed by single architecture decisions. RQ2 applies to finding candidate solutions based on the problems identified in the previous step. Finally, the aim of RQ3 is to discover how students make choices between the candidate solutions and how they evaluate their choices with respect to previously made decisions. It is noted that the requirements engineering activity, though closely related, was performed before the architecting process and is therefore out of the scope of this study. An initial set of requirements was made available to the students. Furthermore, the activity of modifying and describing the architecture (see (Jansen, Bosch and Avgeriou 2008)) was omitted, because of time constraints in conducting the study.

3.3.2 Study Design and Execution

To find answers to the research questions, a descriptive survey (Wohlin et al. 2003) was conducted with students from the seventh semester, in a four-year software engineering program of study at the Fontys University of Applied Science in Venlo, The Netherlands. At that time, the students had at least 3 years of object-oriented programming experience from small software development projects within the study program. Some of them had additional experience from side jobs. They had followed two lectures (three

hours in total) specifically on software architecture. The following topics were covered in this course: the 4+1 architectural views (Kruchten 1995), the recommended practice for architectural description of software-intensive systems (IEEE 2000), the concept of architecture decisions mainly using the template by Tyree and Akerman (Tyree and Akerman 2005), and software architectural patterns (Buschmann et al. 1996). In total, 22 students took part, who were divided into 11 pairs.

To produce an architecting experience, we asked the students to create the software architecture of a non-trivial software system (later referred to as *phase one*). Right after that, the students were asked to fill in a questionnaire, in order to report about their individual architecting experiences (*phase two*). The questionnaire was designed and evaluated according to the guidelines by Lethbridge et al. (Lethbridge et al. 2005).

The architecting case, used in phase one, was a document describing architecturally relevant functional and non-functional requirements for an online selling platform comparable to Amazon.com (Amazon.com Inc. 2012). The case study included requirements for user management, selling books, multimedia and other products, searching for products, notification of sellers and buyers. The non-functional requirements included interoperability, availability, performance and security. In total, nine functional and nine non-functional requirements were given. The students were explicitly allowed to supplement or modify the given requirements, for example because of specific trade-offs.

The architecting activity (phase one) in the experiment took 60 minutes. The students were asked to make all necessary architecture decisions and to document the process of decision making in a mind map. The purpose of the mind map (created on flip charts) was to conserve as much reasoning and as many thoughts of the participants during the decision making process, as possible. No architecting method was imposed on them, nor did they have knowledge about any existing systematic approaches. Additionally, the students were asked to document design options and decisions using a decision template, we provided to them. Laptops with an internet connection were allowed to search for arbitrary information, e.g. to find design options like software patterns or technologies. To gather data in phase two of the experiment, a group-administered questionnaire (Trochim 2001) was handed out to the students right after they finished phase one. The students used their documented decisions and the mind map as help to reflect on the process, while answering the questions. The questionnaire contained a mix of structured and un-structured questions. The structured questions had a five-point interval-level response format, also referred to as Likert-scale (Trochim 2001). To mitigate the risk of ambiguous or hard to understand questions, which comes along with questionnaires (Lethbridge et al. 2005), an instructor explained the questions to the participants one by one. That way, the students could clarify questions before answering. Table 3.1 shows a mapping of the questions in the questionnaire to the research questions formulated in Section 3.3.1. Some questions have a relation to more than one research question; in these cases, the bold-faced 'X' denotes the most relevant research question (except for Q16 and Q17, where all three research questions are equally relevant), while a capital 'S' denotes a secondary research question. Ad-

Table 3.1: *Question Mapping*

Code	Question	RQ1	RQ2	RQ3
Q1	Have you understood and considered the given requirements?	**X**		
Q2	Have you reasoned about the most challenging requirements?	**X**		
Q3	Have the quality attribute requirements played a prominent role during the design?	**X**		
Q5	Have you considered alternatives for the decisions you made?		**X**	
Q6	Have you relaxed requirements to have more design options?	S	**X**	
Q7	Have you thought about the pros and cons of each alternative that you have considered?			**X**
Q9	Have you preferred well-known solutions rather than searching for better alternatives?		**X**	
Q10	Have you sometimes made multiple decisions at the same time?		S	**X**
Q11	Have you rejected decisions?			**X**
Q12	Have you made trade-offs, while making decisions, between multiple requirements?	S		**X**
Q13	Have you come across dependencies between decisions?		S	**X**
Q14	How long did it take since you had a first architectural vision in mind?			**X**
Q15	Does the final architecture significantly differ from your initial vision?			**X**
Q16	How have you come from one decision to the next decision?	**X**	**X**	**X**
Q17	What has gone on in your head when you have thought about the architecture?	**X**	**X**	**X**

ditionally, two more questions were asked, which do not directly map to the research questions: "Do you have the skills to design and program the given system?" (Q4) and "Are you confident that your decisions and the resulting design are sound?" (Q8).

The participation in the study was mandatory. The students received grades for the architecture documentation on the flip charts. It was clearly communicated to the students that the answers in the questionnaire in phase two were not taken into consideration for the grading. This issue will be further discussed in Section 3.5.1.

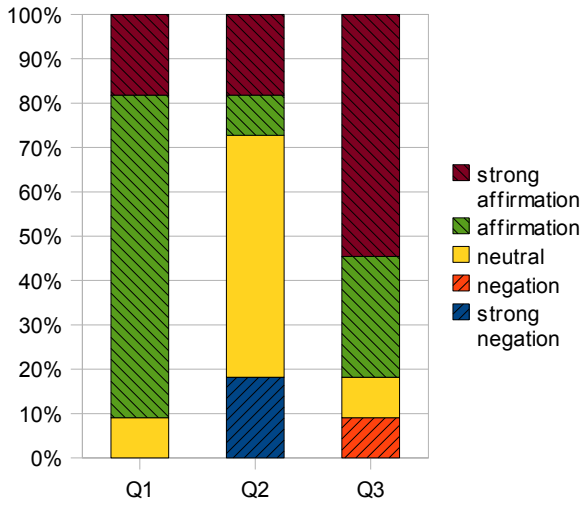


Figure 3.1: *Cumulative frequencies of answers to questions related to RQ1*

3.4 Analysis

We use descriptive statistics to visualize the collected data in the analysis. This section contains one subsection for every research question. Subsection 3.4.4 presents results concerning all three research questions. There are 11 valid data points for each question, one for each student pair.

3.4.1 RQ1 - Architectural Analysis

Questions Q1,Q2 and Q3 from the questionnaire are primarily related to the treatment of architecturally relevant requirements during architectural synthesis. Figure 3.1 shows a stacked bar chart presenting cumulative percentage frequencies of answers to the respective questions. The vast majority of the participants ($> 90\%$) affirmed that they understood and considered the given requirements (Q1). The median answer was 'affirmation'. The answers to question two, concerning the reasoning about the most challenging requirements, do not show a clear trend (Q2, median 'neutral'). More than 80% affirmed that quality attribute requirements played a prominent role during the design (Q3, median 'strong affirmation').

3.4.2 RQ2 - Architectural Synthesis

Figure 3.2 shows the frequencies of answers to questions Q5, Q6 and Q9, related to finding candidate solutions during architectural synthesis (RQ2). More than 70% of the participants affirmed that they considered alternatives for the decisions they made (Q5, median 'affirmation'). The majority of participants did not relax requirements to have more design options (Q6, $> 90\%$, median 'strong negation') and, without any negations, more than 70% of the participants affirmed that they preferred well-known solutions

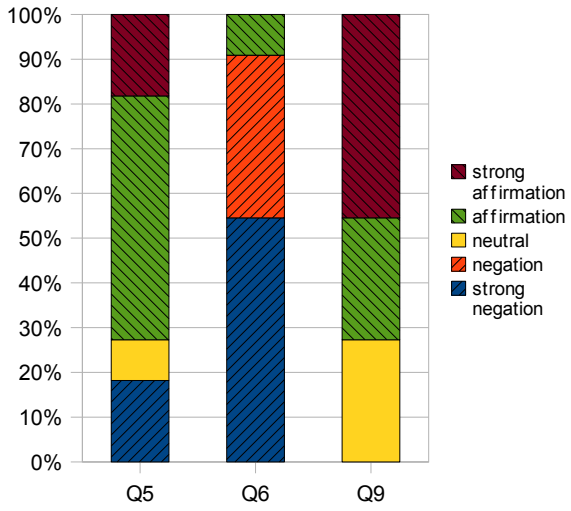


Figure 3.2: *Cumulative frequencies of answers to questions related to RQ2*

rather than searching for better alternatives (Q9, median 'affirmation').

3.4.3 RQ3 - Architectural Evaluation

Questions Q7, Q10-Q15 refer to the choices between candidate solutions and the evaluation of the choices with previously made decisions. Figure 3.3 shows the frequencies of answers. The answers to Q7, referring to the consideration of pros and cons of alternative solutions, show a clear tendency towards affirmation (median 'affirmation'). Q10, related to making multiple decisions at the same time, does not receive a clear result. Although the most frequent answer was 'affirmation', the median answer was 'neutral'. 100% of the participants negated the question about rejecting decisions (Q11, median 'strong negation'). The students also did not consciously make trade-offs between requirements (Q12, $> 60\%$, median 'negation'). The answers to question 13, concerning dependencies between decisions, do not show a clear tendency (median 'neutral'). Question 14 did not have predefined answers. The participants were asked how long it took in minutes since they had a first architectural vision in mind (Q15 refers to this vision). On average, the participants took 13.36 minutes for a first vision. The standard deviation is 9.067 (min: 5min, max: 30min). Finally, without a single affirmative answer, more than 70% negated that the final architecture significantly differed from the initial architectural vision (Q15, median 'negation').

3.4.4 Open questions concerning the whole architecting process

Besides structured questions, we asked the participants to answer two open questions (Q16 and Q17) that concern all three research questions.

In question 16, we asked the students to describe how they got from one decision to the next decision. Four of the pairs stated that they made decisions along the re-

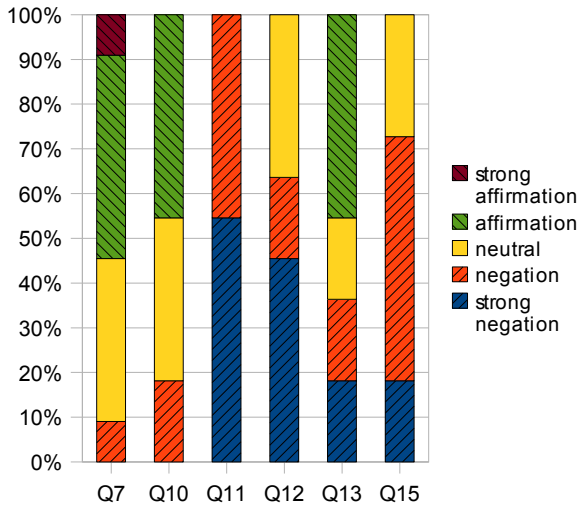


Figure 3.3: *Cumulative frequencies of answers to questions related to RQ3*

quirements (e.g. "Reading requirements one by one"). Two groups mentioned that they used common combinations of technologies as orientation in the decision making process (e.g. Spring as web framework, then Hibernate as object-relational mapper), one group explicitly stated that they first created a list of things to be decided and then made the decisions one by one.

In question 17, the students were asked to freely describe what went on in their heads when they thought about the architecture. The following workflow of decision making can be derived from the given answers: Analyze requirements, find candidate solutions based on own experience, search for alternative solutions, evaluate pros and cons of all candidate solutions, make decision. Exemplary verbatim answers are: "We started with own knowledge and experience, then we thought about alternatives and made pros and cons lists.", "We thought about what was necessary to fulfill requirements, we thought about known technologies, we tried to find some alternatives for these", "Based on the requirements we think about the decisions to take. Then we think of known solutions/technologies and research on further solutions. Finally, we evaluate the different possibilities and make the decisions".

3.5 Interpretation

In this section, the behavior of the students is interpreted and compared to existing approaches in the architecture literature. The section is organized according to the three architecting activities. Findings on Q16 and Q17 that concern all three activities are mentioned where appropriate.

Architectural Analysis

Architectural analysis involves articulating (Hofmeister et al. 2007) and scoping down (Jansen, Bosch and Avgeriou 2008) architecturally significant requirements (ASR). The quality attribute requirements play a prominent role in this activity (Bass et al. 2003, Hofmeister et al. 2009). Usually the ASRs are further prioritized (Jansen, Bosch and Avgeriou 2008) to identify key issues or problematic requirements (Hofmeister et al. 2009, Kruchten 2004b) that require special attention, because they are critical for the architecture. They sometimes become risks (Kruchten 2004b).

The analysis of the students' results showed that most of them intuitively followed these activities. They tried to understand and consider the ASRs and put emphasis on the quality attribute requirements. The only discrepancy is that many students did not identify the most challenging requirements, nor did they prioritize them. It is noticeable that the students do not seem to be aware of risks and consequently do nothing to mitigate them. However, two student pairs strongly affirmed that they did think about the most challenging requirements. A correlation analysis (Kendall's tau) showed that students who affirmed the statement also had strong confidence in the soundness of their resulting designs (Q8) (corr.-coefficient 0.618, sig. 0.023), which allows the conclusion that risk assessment leads to higher confidence in the quality of the architecture.

Architectural Synthesis

Architectural synthesis is the process of finding candidate architectural solutions that (partially) address the distilled ASRs (Hofmeister et al. 2007, Jansen, Bosch and Avgeriou 2008). This activity requires the architect to identify and distill relevant knowledge from own experience and external knowledge repositories (Hofmeister et al. 2007, Tang and Lago 2010). To have more design options, it is sometimes advisable to relax requirements that put too many constraints on possible solutions (Tang and Lago 2010).

The students affirmed that the identification of design options was driven by the requirements. However, they did not relax requirements to have more design options and they also declared that they preferred well known solutions over unknown alternatives. Also, they did not seem to be aware of limitations and constraints that solutions impose on other decisions. Their answers to the open questions reflect that the requirements were used as a kind of checklist to ensure that all of them were covered by at least one solution, without taking into account the relationships and dependencies between decisions. A similar behavior was observed for practicing architects by Clerc et al., who state that the architects' workflow follows a linear approach that satisfies the requirements sequentially (Clerc et al. 2007).

Architectural Evaluation

During architectural evaluation, the candidate solutions are weighed against the ASRs (Hofmeister et al. 2007) to make a design decision. Therefore, the pros and cons of each

design option have to be considered (Jansen, Bosch and Avgeriou 2008, Tang and Lago 2010). Choosing solutions can entail making trade-offs (Hofmeister et al. 2007, Jansen, Bosch and Avgeriou 2008) between requirements. This activity also involves identifying and documenting constraints that decisions impose on future decisions (Bass et al. 2003). Evaluation further ensures that a decision does not violate previously made decisions. Therefore, the architecture is regularly evaluated as a whole after a few iterations (Hofmeister et al. 2009). Some approaches emphasize the need of risk assessment during architectural evaluation (Kruchten 2004b, Tang and Lago 2010) to ensure that no hidden assumptions or constraints behind decisions exist and to assess if additional risks are introduced by a decision.

The study shows strong deviation of the students' behavior from these activities. Although they weighted pros and cons for the design options, they did not consciously make trade-offs between requirements and also neglected to validate the decisions against each other. This explains why the students did not reject decisions. They do not seem to be aware of dependencies and relationships between architecture decisions. Only few students stated that they came across dependencies. In line with these observations, the students quickly came up with a first architectural vision ($13mins$) and did not significantly deviate from this vision any more. This is another indicator that students do not critically evaluate their decisions. This is not very surprising. As mentioned in Section 3.2, Clerc et al. (Clerc et al. 2007) found out that even practicing architects do not regard risk assessment and requirements trade-off analysis as particularly important.

Additionally, we observed that no clear statement was made about the question if they made multiple decisions at the same time. Some students described that they used a kind of reference architecture they knew from comparable projects as a basis, others started from scratch and made decisions strictly sequentially. A correlation analysis (Kendall's tau) showed that students who made multiple decisions at the same time also relaxed requirements to have more design options (corr.-coefficient 0.584, sig. 0.045).

3.5.1 Threats to validity

In this section, possible limitations of the study are presented by discussing internal validity, construct validity and external validity (Kitchenham et al. 2002, Shull et al. 2008).

With respect to internal validity, the questionnaire design and the fact that an instructor verbally explained the questions before they were answered, ensured that the questions were unambiguous and focused on the research questions. The fact that the study was done as a classroom assignment introduces a potential risk. The students received grades for the performance in phase one of the study. Although the questionnaires were not taken into consideration for the grading, some students might have tried to impress the lecturer by giving specific answers. This risk, however, is considered rather low: no evidence in favor of it could be found in the results; and it was not possible for the students to determine which answer would be rated positively or

negatively.

Concerning construct-validity, the fact that only one specific architecting experience was used as a basis for the study introduces the risk that the cause construct was under-represented. The architecting process could be different for other architecting case studies. In this study, the students already had experience building simple web applications. In totally unknown domains, they would have been forced to uncover design options they did not know before. However, the risk is regarded as rather low as working in unknown domains is unrealistic especially for inexperienced designers. It can further be assumed that the architecting process for the used system is representative for those of large and medium-size software projects. We also used multiple variables to cross-check the results concerning the research questions. The risk of researcher's bias was mitigated for the most part, as the structured questions with pre-defined answers do not leave space for interpretation. However, some open questions do exist that were interpreted by the researchers.

With respect to external validity, the subject population in the study might not be representative for the larger population of inexperienced software architects. The participants of the study were undergraduate students in the last year of a software engineering study program. Their state of knowledge is comparable to the lowest level of architecture knowledge that software engineers in practice have. Thus, it can be assumed that this risk is mitigated.

The instrumentation used in phase one of the study might have been unrealistic or old-fashioned. This risk was mitigated by creating a working environment that corresponds to those of practicing architects. The students were allowed to use laptops with internet connections without any restrictions and they could discuss all issues with their partners. In real software projects however, additional constraints (e.g. time, cost, corporate culture, politics) exist that can hardly be simulated in a classroom environment.

3.6 Conclusions and Future Work

To gain insights into the innate reasoning processes of students during architectural design, we conducted a descriptive survey with software engineering students. The architecting process the students followed was compared to existing architecture practices in the literature.

The comparison showed that the students' activities during architectural analysis mostly match with the activities advocated in existing architecture approaches. However, during architectural synthesis and architectural evaluation large discrepancies were observed. As pointed out, some of these were also observed in studies with professional architects, which leads to the conclusion that the problems do not only result from the low level of experience. To move towards a systematic reasoning process, we list the areas that need to be improved and invite the research community to work on providing the necessary methodological and tooling support:

- Prioritize requirements (Jansen, Bosch and Avgeriou 2008) and identify risks in

terms of the most challenging requirements (Hofmeister et al. 2009, Kruchten 2004b) that are hard to fulfill.

- Relax requirements to have more design options, where required (Tang and Lago 2010).
- Search for alternatives, even if known solutions exists that seem to solve the design issue.
- Document why one option was chosen over another one (Tang and Lago 2010) to ensure that design options were not only chosen because of personal bias towards known solutions.
- Reason about possible limitations and constraints that solutions impose on future decisions (Bass et al. 2003).
- Actively consider relationships and dependencies between decisions (Hofmeister et al. 2009, Jansen and Bosch 2005).
- Identify situations, in which decisions cannot satisfy two requirements at the same time. Try to find optimal trade-offs between the requirements (Hofmeister et al. 2007, Jansen, Bosch and Avgeriou 2008, Kazman et al. 2000).
- Determine constraints that decisions impose on future solutions (Bass et al. 2003).
- Assess and actively mitigate risks throughout the architecting cycle (Kruchten 2008).

We hypothesize that systematic support in these areas can verifiably improve the reasoning process. As mentioned in the introduction, we also conducted a study to understand the reasoning practices of successful architects, in order to derive an ideal reasoning process. This study is presented in Chapter 4.

3.7 Acknowledgements

We would like to thank the students from the 2009 course on the Java Enterprise Edition (JEE) at the Fontys University of Applied Sciences Venlo for taking part in the study.

Based on: U. van Heesch and P. Avgeriou – *"Mature Architecting - A Survey about the Reasoning Process of Professional Architects"*, Proceedings of the Ninth Working IEEE/IFIP Conference on Software Architecture, pp. 260-269, 2011.

Chapter 4

Mature Architecting - understanding the decision-making process of architects

Abstract

Architecting is to a large extent a decision-making process. While many approaches and tools exist to support architects during the various activities of architecting, little guidance exists to support the reasoning part of decision-making. This is partly due to our limited understanding of how professional architects make decisions. We report on findings of a survey conducted with 53 industrial software architects to find out how they reason in real projects. The results of the survey are interpreted with respect to the industrial context and the architecture literature. We derive reasoning best practices that can support especially inexperienced architects in optimizing their decision-making process.

4.1 Motivation

A software architecture is the result of a complex system of inter-dependent architectural design decisions (van der Ven, Jansen, Nijhuis and Bosch 2006, Jansen and Bosch 2005). These decisions are made by architects who strive towards an optimal balance between the forces acting on the decisions, including financial and technical constraints. Architecture decisions are the corner stone for the whole software architecture and as such they are vital for the achievement of the system's key drivers and goals.

Architecture decisions are made during the iterative and incremental process of architecting. Hofmeister et al. derived three general, recurring architecting activities, which are common in five industrial architecture approaches (Hofmeister et al. 2007): *Architectural analysis*, which is concerned with identifying architecturally significant requirements (ASR) from a set of architectural concerns and the business context; *architectural synthesis*, which concerns finding candidate solutions for the ASRs; and finally *architectural evaluation* in which decisions are made and validated against the architecture as a whole. These three activities are iteratively performed by moving back and forth between the problem and the solution space (Nuseibeh 2001).

Various approaches have been proposed to support the three architecture design activities. They are either concerned with the architecting process as a whole, or they focus on one of the three activities. Well known examples of the former category are the five processes used as reference in Hofmeister et al.'s general model of architecture design (Hofmeister et al. 2007): RUP, ADD, Siemens' 4 Views, BAPO and ASC.

The latter category includes approaches for architecture evaluation like ATAM, SAAM, or CBAM (Bass et al. 2003); approaches for architecture analysis like the goal-oriented paradigm (e.g. (Van Lamsweerde 2001)); and various methods supporting architects in identifying candidate solutions during architectural synthesis, e.g. architectural patterns (Buschmann et al. 1996), styles (Bass et al. 2003) and reference architectures (Muller 2004).

All of the aforementioned approaches, however, either ignore the reasoning process behind decision-making, or take design decisions into account only as input or output for individual architecture activities (ATAM for instance evaluates the role of design decisions in quality attribute scenarios). To the best of our knowledge, there is no holistic reasoning process that includes all three major architecture activities (analysis, synthesis and evaluation); nor can one be derived from the combination of multiple approaches, as the whole is more than the sum of the parts. In fact, with a few exceptions (e.g. (Tang et al. 2006, Tang and Lago 2010, Tang et al. 2008)), very little research has been done on the reasoning part of decision-making so far.

Design reasoning is a logical process that designers follow when developing architectural solutions (Tang et al. 2008). It applies to all three architecture activities and allows for systematic and disciplined decision making, based on argumentation instead of intuition. Furthermore, if the output of reasoning is documented, it can support stakeholders who were not involved in the decision making process to comprehend decisions and the resulting design. The lack of such reasoning processes, forces software architects to follow an ad-hoc, creative process (Brooks 2010, Tang, Aleti, Burge and van Vliet 2010) relying heavily on their personal experience and expertise. As a consequence, rather inexperienced software architects go through a long and painful succession of sub-optimal decisions, before they can successfully reason about the design options and make informed, well-balanced trade-offs. Training practitioners to follow a systematic reasoning process could narrow the gap between expert architects and novice ones.

In our previous work, reported in Chapter 3, we started analyzing the reasoning process that inexperienced architects follow when they are architecting (van Heesch and Avgeriou 2010). Our aim was to establish a baseline reasoning process that is based on common sense instead of experience. In this chapter, we present the results of a descriptive survey that we conducted with 53 industrial software architects from end-October 2010 until mid-January 2011. We investigate how experienced architects reason in the context of industrial projects and interpret the data according to the industrial context and theory from the literature. Eventually we refine the findings and summarize them into a set of reasoning best practices that junior architects can use to improve their reasoning skills.

The rest of this chapter is organized as follows. Section 4.2 presents related work. In Section 4.3, the design of the study is introduced. The next section presents the analysis of the results, which are interpreted in Section 4.5. The chapter ends with conclusions and directions for future work.

4.2 Related work

Our research is related to three areas within software architecture: architecting processes, architecting practice in the industry and design reasoning.

In order to study the reasoning process, we use the general model of architecture design by Hofmeister et al. as a reference process (Hofmeister et al. 2007). This model consists of three main architecture activities from industrial approaches, namely architectural analysis, architectural synthesis and architectural evaluation. Jansen et al. adopt the model to describe architecture activities from the perspective of architecture decision making (Jansen, Bosch and Avgeriou 2008). They suggest that architecture decisions are the result of a decision-making process comprised of the activities defined in Hofmeister et al. 's general model. Our work is complementary to these approaches, as we explicitly focus on the reasoning process related to each of the architecture activities when making decisions.

The role and duties of software architects in the industry have been analyzed in multiple studies (Clerc et al. 2007, Hoorn et al. 2011, Kruchten 2008, Clements et al. 2007). Findings include that risk assessment and architecture evaluation is not regarded very important by practicing architects and that architects mainly follow a non-iterative approach that subsequently satisfies requirements (Clerc et al. 2007). Hoorn et al. refine those findings, stating that auditing and quality assurance activities are regarded more important with increasing years of experience (Hoorn et al. 2011). Clements et al. suggest that evaluation and analysis are regarded less important in practice than in the literature (Clements et al. 2007). In this study, we also observe the behavior of practicing architects in the context of industrial projects. However, the emphasis in the aforementioned papers is to find out what architects do, i.e. which activities they follow while they are architecting. In our study, we try to understand *how* architects perform the activities in order to derive reasoning practices.

As pointed out in Section 4.1, little work has been done in the field of design reasoning in software architecture. Tang et al. look at design reasoning from a more general perspective, not only specific to software architecture and also take psychological aspects into consideration to explain human behavior during design activities (Tang, Aleti, Burge and van Vliet 2010). In earlier work, they declared the importance of design reasoning in software architecture (Tang et al. 2008, Tang et al. 2006, Bu et al. 2009). The results were used by Tang and Lago to describe an initial set of design reasoning tactics that can be used by software architects to improve their reasoning process (Tang and Lago 2010). Our work also emphasizes the importance of reasoning processes in software architecture. As opposed to Tang et al., who look at design reasoning from a very general, cognitive perspective, our aim is to understand and describe concrete reasoning practices within the three architecting activities found by Hofmeister et al. (Hofmeister et al. 2007) that can be used as guidelines for inexperienced architects.

4.3 Design of the study

4.3.1 Goal

The goal of this survey is to understand the reasoning process that industrial software engineering practitioners follow while they are architecting. To make the research goal concrete, the reasoning process is mapped onto the general model of architecture design by Hofmeister et al. (Hofmeister et al. 2007). The three activities in the model are iteratively performed by architects when making decisions. We aim at understanding the reasoning practices behind these activities, i.e. **how** each of the three activities is performed, which leads to the following research questions:

RQ1 : How do software architects scope and prioritize the problem space during architectural analysis?

RQ2 : How do software architects propose solutions during architectural synthesis?

RQ3 : How do software architects choose among solutions during architectural evaluation?

Research question one considers the involvement of architects in requirements engineering activities such as: requirements elicitation, evaluation of the importance and prioritization of quality attribute requirements and functional requirements and the definition of concrete problems that are small enough to be addressed in the architectural synthesis. The aim of research question two is to find out how architects search for and choose design options based on the output of the architectural analysis. Finally, research question three applies to the assessment of candidate solutions and the evaluation of the architecture as a whole during architectural evaluation. The scope of this question includes architecture reviews and risk management.

4.3.2 Subjects and sampling

The population under study are industrial software engineering practitioners, who have been working in the industry for at least five years and who have been responsible for software architectural design for at least two years. As an additional constraint, subjects were excluded from the study if their daily tasks do not include at least one of the following: requirements engineering, system architecture/design, or software design and specification. To evaluate, if the subjects fit into the target population, we asked the questions shown in Table 4.1. To find appropriate subjects, we used chain referral sampling (also known as snowballing) (Mack et al. 2005): the authors asked professionals from their own network to forward the participation request to other professionals who fit the sampling requirements. In total, 53 people took part in the survey, out of which the results from seven people were excluded, because they did not satisfy the sampling requirements. On average, the remaining participants have worked 18.22 years in the

Table 4.1: *Questions for sampling*

Question	Response format
How many years have you been working as an IT professional?	Positive natural numbers including zero
How many years have you been working as a software architect / designer?	Positive natural numbers including zero
As an architect / designer, which of the following are your tasks?	Possible answers: • project management • requirements engineering • software architecture • software design and specification • test planning and design • reviewing / auditing • programming • others

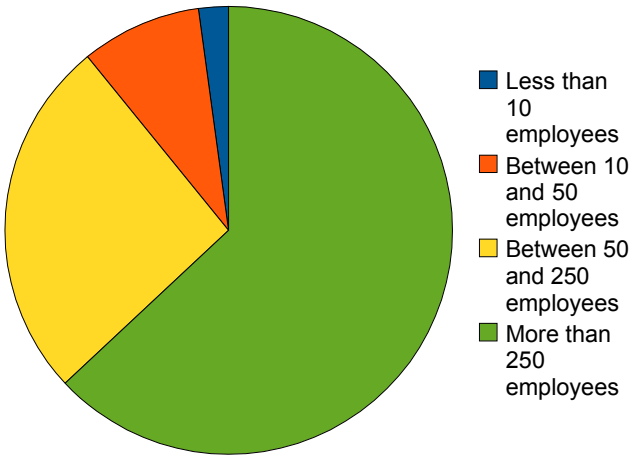


Figure 4.1: *Number of employees in participating companies*

IT-industry (min: 6, max: 35), and on average 10.59 years as a software architect/ designer (min: 4 , max: 30). For statistical means, we asked the participants to specify the numbers of employees in their companies using an interval scale ranging from *less than 10* to *more than 250 employees*. Figure 4.1 shows the distribution of answers. The majority of participants work in large companies.

4.3.3 Data collection

To collect data, a web-based questionnaire was designed with questions that map to the defined research questions. Table 4.2 shows the questions from the questionnaire along with the response format and their relation to the research questions. Some questions have a relation to more than one research question; in these cases, the bold-faced 'X' denotes the most relevant research question (except for Q17 to Q20, where all three research questions are equally relevant), while a capital 'S' denotes a secondary research question.

Using web questionnaires, the subjects and researchers do not have to synchronize in time and place. Participants can fill them in, whenever they find time. A potential disadvantage of questionnaires is that in the case of ambiguous and poorly-phrased questions, there is no interviewer to explain the questions and make sure they are well understood. To mitigate this risk, Lethbridge et al. propose to pilot-test the questions and then re-design those questions that were interpreted wrongly (Lethbridge et al. 2005). We followed this advice and tested the questionnaire initially with one participant from the target population. Right after the questionnaire was filled in online, we had a video conference with the person and asked him to explain how he understood every single question. After this, all questions that were poorly understood were re-designed and we provided additional help texts explaining the questions. Then we repeated the procedure with three additional participants from the target population until every question was explained back to us just the way we aimed it to be understood.

The URL of the questionnaire was sent to the participants by e-mail. It contained a mix of structured and un-structured questions. The structured questions had a five-point interval-level response format, also referred to as Likert-scale (Trochim 2001), whereas the un-structured questions requested numeric input or free-text.

In the questionnaire, we asked respondents to reflect on one specific software project they were involved in as a software architect and which is representative for the way they are working. The whole set of questions in the questionnaire referred only to this concrete project. To focus the participants on this project, we asked them to estimate the project size and specify the domain of the project. The characteristics of the chosen projects are further described in Section 4.4. Furthermore we explicitly requested them to reflect upon their personal thoughts and their personal actions instead of describing their company policies, or what the whole development team did. They were also asked to skip questions they did not understand.

Table 4.2: Mapping of questions and research questions

No	Question	Resp. Format	RQ1	RQ2	RQ3
Q1	How much were you involved in the requirements elicitation of the project?	Likert (Completely to Not at all)	**X**		

Table 4.2 – continued from previous page

No	Question	Resp. Format	RQ1	RQ2	RQ3
Q2	Have you understood the reasoning behind the requirements of the project?	Likert (Completely to Not at all)	**X**		
Q3	Compared to other influencing factors, how important were the requirements as input/motivation for your architecture decisions?	Likert (Very important to Unimportant)	**X**		
Q4	To what extent did you reflect on identifying which of the requirements were hardest to fulfill?	Likert (To the largest possible extend to Not at all)	S		**X**
Q5	How important were the functional requirements for your architectural design?	Likert (Very important to Unimportant)	**X**		
Q6	How important were the quality attribute requirements for your architectural design?	Likert (Very important to Unimportant)	**X**		
Q7	Have you searched for alternative design options, when making decisions?	Likert (Always to Never)		**X**	
Q8	Have you searched for alternative design options even if you already had a solution in mind?	Likert (Always to Never)		**X**	
Q9	Have you thought about the pros and cons of the design options you found?	Likert (Always to Never)		S	**X**
Q10	Have you preferred solutions that you are familiar with, in favor of others that you are not so familiar with?	Likert (Always to Never)		**X**	
Q11	Did you relax requirements?	Likert (Always to Never)	**X**		
Q12	How confident are you that the architecture decisions you made are sound?	Likert (Very confident to Not confident)			**X**
Q13	How often did you decide on multiple architectural solutions at the same time?	Likert (Always to Never)		S	**X**

Table 4.2 – continued from previous page

No	Question	Resp. Format	RQ1	RQ2	RQ3
Q14	How often did you withdraw solutions that you decided on earlier in the project?	Likert (Always to Never)			**X**
Q15	How often did you make trade-offs, while making decisions, between multiple requirements?	Likert (Always to Never)			**X**
Q16	How often did you come across dependencies between architectural solutions you decided on?	Likert (Always to Never)		**X**	S
Q17	How long did it take until you had a first vision of the overall software architecture in mind?	numeric (% of the whole proj. duration)	**X**	**X**	**X**
Q18	Does the final software architecture significantly differ from this initial vision?	Likert (Completely to Never)	**X**	**X**	**X**
Q19	What are the three most important things in decision making for you?	Open	**X**	**X**	**X**
Q20	How have you come from one decision to the next?	Open	**X**	**X**	**X**

4.4 Analysis

We use descriptive statistics and qualitative analysis to describe the collected data. This section is divided according to the research questions. As described in the study design, the participants were asked to reflect on one representative project they had worked on. Table 4.3 shows some characteristics of the chosen projects.

4.4.1 Analysis RQ1 - Architectural analysis

Figure 4.2 shows a stacked bar chart with cumulative frequencies of answers to questions primarily related to RQ1. The colors and hatchings represent the answers to the Likert-scale questions. Depending on the concrete question "1" stands for positive answers (*completely, very important, to the largest extend, always, or very confident*), while "5" represents negative answers (*not at all, unimportant, never or not confident*). Please refer to Table 4.2 for the scalings of the respective questions.

Approximately 60% of the architects stated that they were involved either completely or a lot, in requirements elicitation (Q1). More than 80 % understood the rea-

Table 4.3: *Characteristics of the chosen projects*

Variable	N	Value
Project size in SLOC	19	Min: 50K, Max: 15mill., Med: 400K
Project size in person-months	43	Min: 2, Max: 8000, Med: 150
No of architects involved	46	Min: 1, Max: 100, Med: 3
Domain of the project	46	Top six domains: • Embedded systems (13.04%) • Healthcare (13.04%) • Transportation (13.04%) • Enterprise Computing (10.87%) • Realtime (10.87%) • Telecommunication (10.87%)

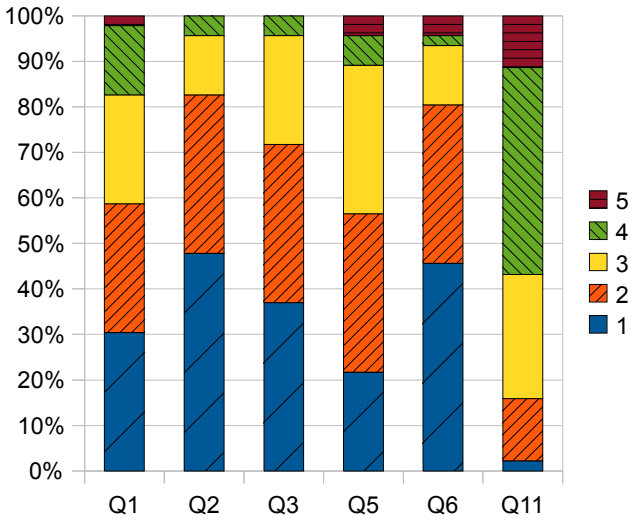


Figure 4.2: *Cumulative frequencies of answers to questions related to RQ1*

soning behind requirements well (Q2). With more than 70% of affirmation, the requirements were regarded important for architecture decisions compared to other influencing factors like technology constraints, budget, or company culture (Q3). About 57% of the participants found the functional requirements important or very important (Q5). The quality attribute requirements were found important or very important by 81% of the respondents (Q6). Finally, the vast majority stated that they seldom or never relaxed requirements to have more design options (Q11).

Apart from the structured questions, some answers to the open question Q19 are related to architectural analysis. The following procedure was used to analyze the open

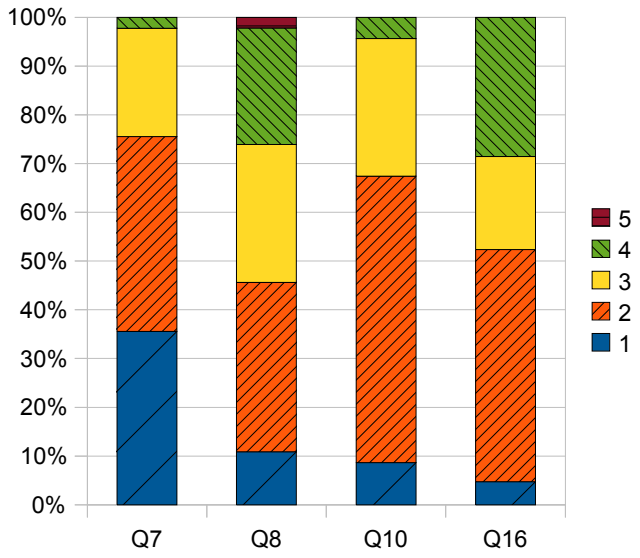


Figure 4.3: *Cumulative frequencies of answers to questions related to RQ2*

answers. We browsed the answers and searched for comments related to the research question. From the comments, we derived single concrete statements expressing what the respective participant answered. Finally, we counted the occurrences of the derived statements.

With respect to RQ1, the following statements were made. We only mention statements that were concordantly made by at least three participants. The numbers in brackets express the number of participants who made that comment: *understand the problem domain* (12 times), *have well-defined requirements* (7 times), *consider non-technical requirements like time and resource limitations, political issues and return on investment* (7 times), *involve stakeholders in the decision making process* (7 times), *regard performance* (4 times), *consider functional- and non-functional requirements equally* (3 times), *negotiate and relax requirements* (3 times). In total, 48 comments were related to architectural analysis.

4.4.2 Analysis RQ2 - Architectural synthesis

Figure 4.3 illustrates cumulative frequencies for Likert-scale questions related to architectural synthesis (RQ2).

With only one exception, all participants (74% plus 22% neutral) indicated that they usually search for alternative design options when making decisions (Q7). Significantly less participants (46%) search for alternative design options if they already have a suitable solution in mind (Q8). The respondents concordantly prefer well-known solutions in favor of unknown alternatives (Q10, 68% affirmation, 4% negation); more than 50% answered that they often come across dependencies between architectural solutions they decide on (Q16).

As for RQ1, we qualitatively analyzed the answers given to Q19 with respect to RQ2. The following statements were made by at least three participants: *Know the solution space* (7 times), *find multiple design options* (7 times), *discuss design options with colleagues*

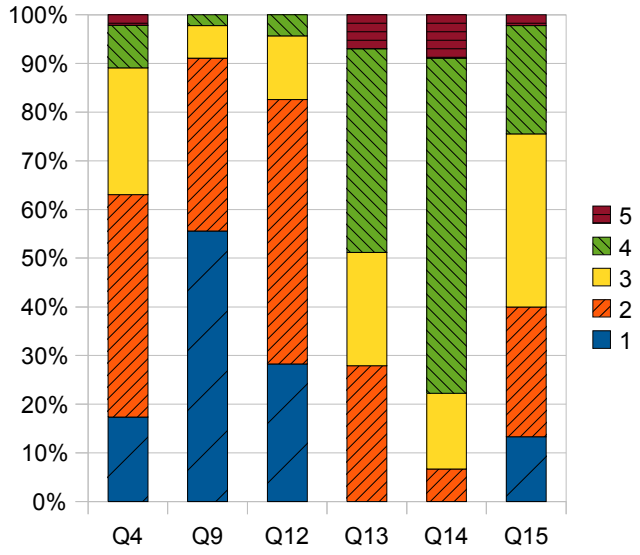


Figure 4.4: *Cumulative frequencies of answers to questions related to RQ3*

(5 times) and *choose the simplest design* (5 times). In total, 31 comments were related to architectural synthesis.

4.4.3 Analysis RQ3 - Architectural evaluation

The results with respect to RQ3 are shown in Figure 4.4.

With roughly 10% of negation, more than 60% strongly reflected on identifying the most challenging requirements (Q4) and thought about the pros and cons for each of the considered design options (Q9). Likewise, more than 80% had strong confidence in the soundness of their decisions (Q12). The question if multiple decisions are at the same time was answered less clearly. Approximately 50% negated the question, while less than 30% stated that they usually make multiple decision at the same time (Q13). With strong significance, the vast majority of the respondents (76%) did not withdraw decisions they decided on earlier in the project (Q14). Q15 does not show a clear tendency. The mode answer was neutral with a tendency towards affirmation (39% compared to 24% negation).

Concerning Q19, the following statements were made by at least three participants: *Understand pros and cons of each design option* (7 times), *validate decisions in reviews* (3 times). In total, 16 comments were related to evaluation.

4.4.4 Analysis of questions 17,18 and 20 in the questionnaire

Questions 17,18 and 20 in the questionnaire cannot be clearly assigned to a specific research question. We asked the participants how long it took, relative to the whole duration of the architecture phase, until they had a first vision of the architecture in mind (Q17), how much they derived from this initial vision after having completed the architecture (Q18), and how they came from one decision to the next (Q20).

Forty-four answered question Q17. On average, it took the architects 17.2 % (min: 5%, max: 75%, med: 12.5 %) of the time spent on architecture to develop a first vision of the overall system. The same number of people answered question 18. The mode answer to this Likert-type question was "moderately" (39%), 11% answered that the final architecture differed from the first vision completely (2%) or a lot (9%). The remaining 46% answered, that the final architecture differed slightly (35%) or not at all (11%).

The open answers to Q 20 describe the overall process that architects follow. For the qualitative analysis of the answers, we use the same procedure as for Q19, i.e. we derive statements from the answers and count the occurrences of every statement.

The following statements were made by at least two participants. The number in brackets is the number of occurrences of the respective statement: *requirements should be prioritized. The important ones should be regarded first* (6 times), *architecture is iteratively refined and improved* (6 times), *there is no specific order in decision making* (6 times), *the requirements guide the decision making process* (5 times), *some decisions have to be made in combination* (3 times), *sometimes, candidate solutions should be prototyped to find the right one* (3 times), *some decisions have strong dependencies* (3 times), *decisions from other projects can guide the decision making process* (3 times), *the decision making process is driven by risks* (2 times).

4.5 Interpretation

In this section, we interpret the findings from the analysis. Specifically, we interpret the architects' answers and compare them to existing approaches in the software architecture literature. The section is organized according to the three architecting activities.

4.5.1 Architectural analysis

The purpose of architectural analysis is to define and scope the problems that have to be solved by the architecture (Hofmeister et al. 2007, Jansen, Bosch and Avgeriou 2008). The outcome of this activity is a set of architecturally significant requirements that serve as input for the architectural synthesis.

The analysis of RQ1 showed that practicing architects are usually involved in the requirements elicitation of the project (Q1); this means that they do not just receive requirements and constraints as artifacts from requirements engineers, but they are actively involved in the communication with customers. This differs from architecture approaches in the literature, which generally assume that a set of requirements is given to the architects as input for the architectural design (see for instance ADD (Bass et al. 2003)).

The involvement in requirements elicitation partially explains the results of Q2: the vast majority of architects stated that they understood the reasoning behind requirements very well. The answers to Q19 also showed that architects find it very important to understand the problem domain and have well defined requirements. These state-

ments were the most frequent answers to Q19, which allows the conclusion that a deep understanding of the requirements and the problem space is regarded as essential by most industrial architects.

Requirements are an important input factor for architecture decisions (Q3). This includes functional and quality attribute requirements (Q5, Q6), although the quality attribute requirements are clearly found more important than the functional requirements. Apart from the functional and quality attribute requirements the architects mentioned non-technical concerns like time and resource limitations, political issues and return on investment as important drivers for architecture decisions. This is comprehensible, as industrial practice is constrained by factors like budget and time limitations, development teams being experienced in specific technologies and customers who indicate the use of specific software systems, because of in-house software licenses. Consequently, this means that an architecture that perfectly fulfills the functional and quality attribute requirements is not necessarily the right architecture in every organizational context. This finding stresses the need to document the rationale of design decisions, as decisions influenced by non-technical concerns may seem irrational or at least incomprehensible to stakeholders who are unfamiliar with those concerns. In Chapter 7, we describe the concept of decision forces, which we promote as first class entities, to capture all of these impacts on architecture decisions.

Most of the architects answered that they seldom relax requirements (Q11). This sounds surprising in the first place, as relaxing requirements that are too constraining would be a means to get more design options (Tang and Lago 2010). However, it is in accordance to the answers to Q1 and Q2: architects who are highly involved in the requirements engineering activity for a project gain deep knowledge about the problem space and have presumably already ensured that the requirements are not unrealistic or too challenging. Nevertheless, three architects mentioned that negotiating and relaxing requirements is one of the most important activities (Q19). A correlation analysis showed, that the three architects who made this comment were less involved in the requirements elicitation (med: 3 compared to med:2), understood the requirements worse than the average (med:3 compared to: med:2) and relaxed requirements more often than the average(med:3 compared to med:4). This means that architects who are less involved in the requirements elicitation process have to relax and negotiate requirements more often later.

4.5.2 Architectural synthesis

Architectural synthesis is the main activity in architectural design as it is concerned with identifying candidate solutions for the architecturally significant requirements (Hofmeister et al. 2007). Architects have to make use of their existing design knowledge, or create new knowledge by consulting external knowledge repositories (Hofmeister et al. 2007, Tang and Lago 2010) in order to find candidate solutions. Tang et al. suggest that creative design requires architects to refine and formulate the problem and solution space at the same time (Tang, Aleti, Burge and van Vliet 2010), in line with the “Twin

Peaks" model (Nuseibeh 2001). This implies that architectural analysis and architectural synthesis are closely coupled activities.

The architects who took part in the study very frequently searched for multiple design options when making decisions (Q7). However this happened significantly less often if the architects already had a solution in mind (Q8). This might be due to the fact that searching for design options is an effort-intensive task, for which designers often do not afford the resources to perform adequately. Furthermore, designers need to search for design options on external knowledge repositories and choose candidate solutions based on unproven assumptions. In line with this finding, architects prefer solutions they are familiar with, instead of unfamiliar alternatives (Q10). It is less risky to select known solutions, even if they have known shortcomings, because these can be assessed and mitigated. Unfamiliar alternatives require substantial effort to reflect on and analyze, which is not always possible within the tight budget of a project.

The analysis of Q19 showed that architects find it very important to know the solution space and have multiple design options. This supports the finding by Cross (Cross 2004) and Tang (Tang et al. 2008), who concordantly found that designers create better designs when they explicitly take multiple design options into consideration. In cases where the participants of our study did not have enough knowledge about the solution space to find candidate solutions on their own, they stated that they discussed design options with colleagues. They also emphasized that the simplest design for a problem should be chosen. This indicates that the concerned systems have such a great size and complexity that simplicity of design solutions is of paramount importance to manage this complexity.

The answers to the open question Q20 and to the structured question Q 16 reflect that architects are aware of dependencies that exist between some of the decisions. Three architects suggest that dependencies have to be considered when finding candidate solutions. In addition, some architects consider that certain dependencies are so significant, that the related decisions can only be made in combination. The analysis of dependencies between decisions is not supported in current architecting processes (Hofmeister et al. 2007), but it has been discussed extensively within the architecture knowledge community (Jansen et al. 2009).

4.5.3 Architectural evaluation

During architectural evaluation, the candidate solutions from the synthesis activity are validated against the architecturally significant requirements (Hofmeister et al. 2007). This entails considering advantages and disadvantages of the candidates (Jansen, Bosch and Avgeriou 2008). Some of the candidate solutions require trade-offs to be made between multiple requirements (Hofmeister et al. 2007, Tang and Lago 2010). Dependencies between decisions and constraints for future decisions should be analyzed and documented thoroughly (Bass et al. 2003). Many architecture approaches regard risk assessment as integral part of architectural evaluation, e.g (Tang and Lago 2010) and (Kruchten 2004b). Finally, the architecture as a whole should be evaluated regularly to

make sure that decisions are consistent with each other, e.g. that older decisions do not harm constraints that came up after they were made.

With respect to the evaluation of candidate solutions, almost all architects stated that they usually think about the pros and cons of design options (Q9). Some emphasized the necessity of prototyping different candidate solutions, before making a decision (Q20). They also had high confidence in the soundness of their design (Q12), which indicates that they had made an informed choice with respect to the pros and cons of the design options. Comparably few architects often decide on multiple decisions at the same time (Q13), although the majority of architects was aware of dependencies between decisions (Q16). A correlation analysis (Kendall's tau) did not show a significant correlation between Q13 and Q16 (corr.-coefficient 0.256, sig. 0.066), which means that architects who were aware of dependencies between decisions did not necessarily make more decisions in combination. This may also be due to the complexity of the various problems and their solutions; each design decision may be complex enough in its own right, making it difficult to take into account its dependent decisions.

Regarding Q14, the results are clear: architects seldom reject decisions they made before. This is in line with the findings of Tang et al., who suggest that designers are reluctant to changing their minds (Tang, Aleti, Burge and van Vliet 2010). This, however, could indicate that previously made architecture decisions are seldom revisited, i.e. the architecture is not validated at the end as a set of decisions. One comment to Q19 is a strong affirmation of this attitude: "once the decision was made it is not allowed to re-discuss it". This may be again due to the time and budget constraints of the projects: there is simply not enough time and resources to continue reflecting on past decisions; the architects need to consider them finalized and move on.

In general, architecture evaluation seems to be less important for practicing architects than the other two activities. This assumption is supported by the fact that only 16 out of 95 comments to Q19 (the most important thing in decision making) concerned architecture evaluation (31 for architectural synthesis, 48 for architectural analysis). Only three architects mentioned the necessity for reviews. Additionally to this finding, we observed that architects do not seem to pay particular attention to risks. In the answers to Q19, the word "risk" was not mentioned at all. In a survey with Dutch software architects, Clerc et al. also found that risk assessment was not regarded particularly important (Clerc et al. 2007). However, some of our results show that architects at least unconsciously perform risk mitigation, for instance by reflecting on identifying the requirements that are hardest to fulfill (Q4) and preferring well-known solutions in favor of unknown alternatives (Q10). In question 20, six architects explicitly answered that requirements should be prioritized and that the most important ones should be addressed first, which is also a means to minimize risks.

4.5.4 Overall architecting process

To understand the reasoning followed within the overall decision-making process, we interpret the findings from open question Q 20, in which we asked the participants to

describe freely how they come from one decision to the next. One of the most frequent comments was that the prioritized requirements guide the decision making process. This, however, does not imply a sequential approach to decision making. Instead, many architects stated that architecture is iteratively refined and improved, which is in line with architecture approaches in the literature (Hofmeister et al. 2007). The iterative nature of architectural design is also indicated by the answers to Q17 and Q18. Architects rather quickly develop a first vision of the overall architecture (<20% of the time for the complete architecture phase, Q17) and then refine this vision until the architecture is complete without significantly deriving from the initial vision any more (Q18).

As opposed to the architects who used the requirements to imply the order of decisions, the same amount of architects reflected that there was no specific order in decision-making. This is an indication that the decision-making process follows an arbitrary reasoning path; we argue that further research should be conducted to provide practicing architects with effective methods and tools to structure their decision-making sequence.

Finally, only one respondent named a concrete architecture approach he followed (in his case the rational unified process). This could indicate that a great part of the participants does not follow one particular architecture approach from the literature; instead they at least partially adopt architecture activities to define their own customized approach to architecture. This conjecture, however, must be validated in further research.

4.5.5 Threats to validity

To describe the internal validity of empirical results, it is important to exclude, or at least explain confounding variables and other sources of potential bias (Kitchenham et al. 2002). Surveys generally bare the risk of poorly controllable variables (Ciolkowski et al. 2003), at least if online questionnaires are used as a data collection method. In such cases, the only means to control variables is by exclusion or by randomization. In this study we used both: participants who were not sufficiently experienced in software architecture were excluded from the study, and other potential variables were randomized by using snowballing as sampling technique. Other potential threats to internal validity (especially construct validity) in questionnaires are ambiguously and poorly-worded questions (Lethbridge et al. 2005). To mitigate this risk, we pilot-tested our questionnaire in multiple iterations until the respondents understanding of the questions matched our intentions (see Section 4.3 for more details).

An addition threat to internal validity is the fact that the answers to the open question Q19 (the three most important things in decision making) could have been influenced by the structured questions, we had asked before. However, the majority of the answers were complementary to the questions. Few of the answers indeed demonstrate such a correlation, but in these cases, the participants still had to make a choice that reflected their personal behavior.

An additional limitation of questionnaires is the uncertainty, whether or not the participants answer truthfully. We tried to keep this risk low by ensuring the respondents

that no data was gathered that would allow us to draw conclusions with respect to the identity of the respondent. Moreover, if people are not willing to be honest, they usually do not volunteer for such a survey. However, this risk can never be excluded totally.

External validity is the extent, to which conclusions can be generalized and capture the objectives of the study (Kitchenham et al. 2002). It is primarily concerned with the representativeness of the sample for the target population (Ciolkowski et al. 2003). The target population of this study were software architects, who have been working in the industry for at least five years and who have been responsible for software architectural design for at least two years. We presume that our findings concerning the reasoning process can be generalized to the population of architects who fit to these sampling criteria. However, one might argue that the reasoning process is not just influenced by the experience of the architect, but also by the characteristics of the software project (e.g. size and domain) and the culture of the company, in which the project is carried out. The demographics of the participants demonstrate that they worked in a variety of application domains and companies, as discussed in the following two paragraphs.

The influence of the company culture is limited by the fact that multiple companies took part, which were not chosen by us directly. We know of at least eight different companies who took part in the study, because respondents from eight different organizations across Europe and the USA sent us e-mails after participating, to state their interest in obtaining the study results. Data about the domain and size of the project that the architects considered in the study was collected in the questionnaire. The average project size was 1441 person-months (1.4 mill SLOCs), which means that mainly large projects were regarded.

The domains of the project included software engineering (17%), embedded systems (13%), transportation (13%), healthcare (11%), realtime (11%), command and control (9%), enterprise computing (9%), telecommunication (9%), finance (8%), e-commerce (6%) and manufacturing (6%). Thus, a wide range of projects from different domains was covered. To understand the influence that the project domain had on the results, we correlated the domains with the dependent variables (Spearman's rho). At the significance level of 0.05 (2-tailed), the domains finance, transportation and healthcare showed correlations. Architects from the finance domain reflected less on identifying which of the requirements were hardest to fulfill (Q4, corr.-coeff: -.291, sig. 0.05), they spent less effort on searching for alternative design options, if they already had a solution in mind (Q8, corr.-coeff: -.306, sig. 0.039) and had less confidence in the soundness of their decisions (Q12, corr.-coeff.: -.303, sig. 0.041). Architects from the healthcare sector more often searched for alternative design options, if they already had a solution in mind (Q8, corr.-coeff: .307, sig. 0.038). In the transportation domain, architects reflected more on identifying which of the requirements were hardest to fulfill (Q4, corr.-coeff: .298, sig. 0.044) and also thought more about the pros and cons of design alternatives (Q9, corr.-coeff: .296, sig. 0.049). However, the fact that only few correlations were found shows that project domains seem to have no significant influence on the reasoning process.

4.6 Conclusions and future work

We conducted a descriptive survey with industrial software architects from several companies and project domains to get insight in the reasoning process followed during architectural design. The results were interpreted according to the pragmatic constraints in the industry, as well as established architecting approaches in the literature. As explained in Section 4.1, our aim was to define reasoning best practices guiding especially inexperienced architects in the three architectural activities. The following best practices were derived from our results:

- **Architectural Analysis:** A deep understanding of the requirements and the problem space is essential for successful architecting. If possible, architects should get involved in the requirements elicitation to gain a better understanding of the requirements and other architectural drivers like time and budget-constraints. If, for some reason, they cannot get involved in requirements gathering, they should make sure that requirements are not too constraining or unrealistic and eventually negotiate and relax them with the respective stakeholders. Requirements should be prioritized; the most important ones and the ones that are hardest to fulfill should be regarded first, as they bare potential risks. Requirements are an important part of the rationale behind architecture decisions and as such they should be documented adequately.

- **Architectural Synthesis:** It is advisable to search for multiple design options and get to know the solution space well when making decisions. In cases where time and budget is very limited it is sometimes practical to consider less design options, if the architect already has a working solution in mind that has proven itself in prior projects. In cases where multiple design options equally fit to the design problem at hand, it is less risky to stick to a solution the architect knows well. When weighing pros and cons of design options, a colleague can act as a sounding board to make sure that choices are informed and unbiased by personal preference. In cases where multiple decisions have strong dependencies, they can be discussed as a whole, i.e. the total of such strongly-dependent decisions can be treated as a single decision. Finally, as in other design disciplines, simplicity should be a key goal in software architecture; unnecessary complexity should be avoided.

- **Architectural Evaluation:** In architectural evaluation, candidate solutions must be validated against the ASRs to make a decision. In situations, in which a decision cannot satisfy two requirements at the same time, the optimal trade-off between those requirements has to be found. Prototyping design options or combinations of design options can help understanding solutions and provides additional rationale for informed choices. Apart from evaluating design options, the architecture should regularly be evaluated as a whole to ensure consistency between the decisions and to uncover hidden constraints. If this is not possible due to time and

budget constraints, it should at least be done once at the end of the architecture phase. A thorough documentation of architecture decisions can reduce the effort needed for their evaluation.

There is one more best practice that spans through all three activities of architecture design and concerns the iterative refinement and improvement of an architecture. Architects should try to develop an overall vision of the complete architecture rather quickly, and then revisit the constituent parts of the vision to finalize the decisions. Decisions from comparable projects can serve as a starting point to develop the vision and can furthermore help to make sure that no important considerations were forgotten.

In Chapter 6, we present a framework for architecture decisions, which was developed to effectively support software engineers in the different activities of architectural design.

In our previous work, we started analyzing the reasoning process of inexperienced software engineers (van Heesch and Avgeriou 2010) (see Chapter 3). We performed additional studies with graduate students who have followed lectures specifically in software architecture and undergraduate students who have not had any software architecture education. This distinction was made to find out in how far software architecture education influences the way students reason about architecture (Section 10.2.1 elaborates on this study). We assume that students who have had some kind of software architecture training adopt at least some of the practices and methods they were taught, while others are ignored. We plan to use these results and compare them with the findings presented in this article, in order to propose appropriate training material for inexperienced architects.

Acknowledgements

The authors would like to thank all respondents of the survey for their participation. Especially, we thank Philippe Kruchten, Antony Tang, Christian Dietrich and Kevin Erhardt for pilot-testing and discussing the questionnaire with us.

Part III

Modeling architecture decisions

Based on: U. van Heesch, P. Avgeriou, U. Zdun and N. Harrison – *The supportive effect of patterns in architecture decision recovery - A controlled experiment*, Science of Computer Programming, 77(5):555-576, 2012.

Chapter 5
Using patterns in architecture decision recovery

Abstract

The documentation of software architecture decisions is important to help the stakeholders understand the system and the rationale behind architectural solutions. In practice, the documentation of such decisions is regularly done after the fact, or skipped completely. To support software maintenance and evolution, the decisions have to be recovered and described. This is often hindered by the fact that the original architects are not available any more, or they do not completely remember the reasons for making the decisions. Additionally, the whole process is very expensive. In this chapter, we hypothesize that architecture decision recovery can be more efficient by focusing on recovering decisions related to applying architecture patterns. To test this hypothesis, we designed a controlled experiment that was conducted to analyze the impact of architecture patterns on the quality and quantity of architecture decisions recovered after the fact. We are able to provide statistical evidence that a focus on patterns significantly increases the quality of decisions, while no conclusive evidence concerning the quantity of decisions was found.

5.1 Motivation

During the architectural design of a software, many decisions are made that influence the fundamental structure and behavior of the software system to develop. The architects responsible for making these decisions have to take into consideration the concerns of the most important stakeholders, quality attribute requirements, architecturally-significant functional requirements and constraints that limit the potential outcome of the decisions. Architecture decisions satisfy some of the concerns while they may potentially violate others. As a consequence, architecting involves negotiations between stakeholders and making trade-offs between different requirements and concerns that have to be satisfied during the software design. The perfect solution does not exist; the rationale of architecture decisions explains the related trade-offs and optimizations.

While architects consciously and subconsciously make these decisions, they regularly neglect to document them appropriately (Hoorn et al. 2011). In some cases, the outcome of decisions is represented in architecture documentation, various UML design diagrams, or at the very least in the source code; however, the exact problem that is solved, the concerns that were considered, and the rationale behind the decision are usually omitted (van der Ven, Jansen, Nijhuis and Bosch 2006).

After some time, when the project advances, even the architect who originally made the decisions will have difficulties remembering all the details and eventually

the knowledge gets lost to a great extent. In the literature, this problem is called *architectural knowledge vaporization* (Harrison et al. 2007, Hoorn et al. 2011, Jansen, Bosch and Avgeriou 2008, van der Ven, Jansen, Nijhuis and Bosch 2006).

This phenomenon becomes especially problematic when software systems are maintained or extended. During software evolution, developers must understand the existing system well in order to make informed decisions on changes and extensions. New requirements and changing system behavior make it necessary to carefully review the original architecture decisions before making additional ones. In absence of decent project documentation, the architectural knowledge and especially the past decisions have to be recovered. Otherwise, new architecture decisions may conflict or override existing ones, or may repeat past mistakes. Therefore, recovering architecture decisions is important for a successful system evolution.

Unfortunately, recovering architecture decisions presents several challenges. If the project documentation is poor, the recovery of architecture decisions is a resource-intensive task that requires a lot of experience and has a high risk for ambiguity and misunderstandings. If architecture decisions were not explicitly documented by the original architects, the new software development team, responsible for making changes to the system, typically has to rely on the running application, the source code, incomplete textual documentations, end-user manuals, and fragmentary or out-of-date design diagrams (e.g., in UML).

It is often challenging enough to identify architectural solutions and the corresponding decisions on the basis of these artifacts. Finding out why they were chosen, which requirements and concerns they satisfy, and which consequences the implementation of the approaches has, requires vast knowledge and experience from the analysts. To make matters worse, many architectural solutions cannot be understood in isolation; they are pieces of a larger puzzle that only make sense if they are examined in the architecture as a whole.

One way to efficiently recover architecture decisions is to look for patterns applied in the architecture and reuse their extensive documentation, which can be found in the pattern literature, in the context of the system under study. Patterns typically describe the problem space in which they are applicable and give advice for applying the solution they propose. In that respect, architectural patterns capture many important aspects of the decision to apply them in an architecture (Harrison et al. 2007). Once the pattern is identified during decision recovery, the pattern description can be used to explore the pattern's problem space, the consequences of applying it, related decisions, and possible trade-offs the original architects made. Of course, not all architecture decisions are related to applying patterns; but some of the most important ones are. We conjecture that architecture decision recovery based on patterns is more efficient than ad-hoc, intuitive decision recovery.

The goal of this chapter is to empirically validate whether architecture decision recovery is more efficient regarding the quality and quantity of architecture decisions, if the recovery focuses on identifying applied patterns. Specifically, we intend to answer the following research question:

Are the quality and quantity of recovered architecture decisions higher if the recovery focuses on identifying applied architectural patterns than in the general case?

To answer this research question, we conducted a controlled experiment during the European Conference on Patterns Languages of Programs (EuroPLoP) (Hillside Europe e.V. 2009) in July 2009 and during a software architecture workshop for industrial practitioners in Venlo, the Netherlands, in April 2011. In total, 33 software engineering experts from academia and from the industry took part. They were asked to recover architecture decisions on the basis of an architectural documentation of the JBoss J2EE application server (JBoss.org 2012). Half of the participants were explicitly asked to focus on identifying patterns in the architecture, while the other half was told to rely on their experience and intuition when performing the recovery. The data from the experiment was analyzed, and the quality and quantity of the recovered architecture decisions were compared.

The results of the experiment provide strong evidence for the benefits of using patters concerning the quality of recovered decisions. The study did not provide conclusive evidence concerning the quantity of decisions.

The rest of this chapter is organized as follows: Section 5.2 presents related work. Section 5.3 explains the design of the controlled experiment including the introduction of variables and hypotheses, while the next section presents details about the execution of the experiment. We analyze the results of the study and present the hypotheses testing in Section 5.5. Section 5.6 contains an interpretation of our findings, a discussion of threats to validity, and finally observations and lessons learned. Section 5.7 concludes and presents future work.

5.2 Related work

The design of this experiment and the theoretical background of the hypotheses presented in Section 5.4 are related to multiple research areas: software architecture, architecture recovery and software patterns.

Within architecture recovery, we distinguish between architectural reconstruction and architecture decision recovery. The former concerns the reconstruction of an architecture, which was never documented by the architects, or whose documentation is no longer synchronized with the system "as-is" (Bass et al. 2003). The latter mostly focuses on recovering the decisions regarding architectural solutions and the reasoning behind the latter; especially concerning the satisfaction of requirements. It does not only answer the question *what* the architecture is like, but also *why* it is like that. Thus, architecture decision recovery is complementary to architectural reconstruction. The following paragraphs discuss the related work in each of the aforementioned areas.

Many definitions exist for software architecture. In ISO/IEC/IEEE 42010 (ISO/IEC/IEEE 2011), the international revision of IEEE Std 1471-2000 (IEEE 2000), the architecture of a system is defined as "fundamental concepts or properties of a system in its environment embodied in its elements, relationships, and in the principles of its

design and evolution". In the software architecture literature, it is also described as the result of making a set of design decisions that impact the overall structure and behavior of a software system (Bosch 2004, Jansen and Bosch 2005, van der Ven, Jansen, Nijhuis and Bosch 2006). These decisions are usually called architecture decisions. While some approaches mainly document the outcome of these architecture decisions in different architectural views (e.g. (Clements et al. 2010, ISO/IEC/IEEE 2011, Kruchten 1995), others focus on documenting the decisions themselves (Tyree and Akerman 2005, van der Ven, Jansen, Nijhuis and Bosch 2006).

However, these approaches focus on the documentation of software architecture during the architecting process, or at least by the architect himself. They primarily concern forward engineering scenarios and try to conserve knowledge that is, or might become, useful in the future. In contrast to this, during architecture recovery, this knowledge, and often also the people possessing it, are partially or totally unavailable.

Several approaches exist that mainly use source code as a basis for architecture reconstruction (Kazman and Carrière 1999, Krikhaar et al. 1999). Krikhaar et al. propose an approach that uses the source code and naming and coding conventions to extract the architecture of a system ex post (Krikhaar et al. 1999). Although the original architects, if available, can make a contribution to this process, the goal is not to recover rationale, but to expose the architecture of a system in suitable representations to allow impact analyses on quality attributes and to incrementally improve the architecture. Kazman and Carrière present an approach to reconstruct architecture that centers around "Dali", a suite that integrates multiple tools to extract and analyze software architecture (Kazman and Carrière 1999). Again, the approach involves the extraction of possibly multiple models from the source code and other programming artifacts that describe the architecture including elements, relationships, and attributes of relevant entities. These models can be seen as different views on the architecture.

In contrast to the static source code analyzes approaches presented above, dynamic system analysis focuses on the runtime behavior of systems. Yan et al. describe an approach called "DiscoTect" (Yan et al. 2004). The architecture of a running system is analyzed using state-machines to identify common patterns of runtime behavior in monitored system events. As a result, DiscoTect identifies applied architectural styles, whose runtime patterns have been defined in state machines. Other outcomes of the method are different views representing the architecture of the system at runtime.

In all aforementioned approaches, and mainly in all other architecture reconstruction approaches (Koschke 2009), the source code, system events and other artifacts used to configure and run the system are automatically processed using tool support to build different types of models that represent the architecture of the analyzed system as-is. They do not aim at recovering the problem space of the architectural constructs or even the rationale behind them. Architecture decisions are not made explicit.

Jansen et al. present an approach to recover architecture decisions (Jansen, Bosch and Avgeriou 2008). The described method involves reconstructing detailed designs and several architectural views on a level of abstraction that is suitable for recovering architecture decisions. Source code, and information from the original architects, form

the basis for the architecture reconstruction. This approach has a different focus than the previously mentioned ones, as representations of the architecture are solely created for the purpose of recovering architecture decisions. Depending on the purpose of the models, they can remain on a high level of abstraction. In contrast, this chapter focuses on the recovery of architecture decisions based on existing architecture documentation that was created manually and is not necessarily a result of source code analysis. Additionally, the recovery of decisions is done manually and profits from interpretation skills that humans have in contrast to machines. This makes the approach applicable for situations, in which the documentation neither involves multiple architectural views, nor a detailed design of the whole system, but fragmentary textual descriptions that include box-and-line diagrams and other sketches of aspects of the system. Moreover, it explicitly supports the recovery of the problem space and the rationale behind decisions, not just the specifically applied solutions.

Software architectural patterns[1], like all patterns, capture generic solutions to recurring problems in specific contexts (Alexander 1979, Buschmann et al. 1996, Gamma et al. 1995). They provide reusable architectural knowledge for a particular application domain (Schmidt and Buschmann 2003). Architectural patterns reason about design alternatives, consequences, and trade-offs concerning software qualities, which are performed when applying them (Buschmann et al. 1996). Architectural patterns explicitly discuss the consequences of their usage concerning the quality attributes of the target architecture and mention related patterns (Buschmann et al. 1996).

A comparison of patterns and architecture decisions is presented in (Harrison et al. 2007). In this paper, the advantages of documenting patterns applied in a software architecture are discussed. Other approaches exist that also make use of patterns as source of architectural knowledge (Babar and Gorton 2007, van Heesch and Avgeriou 2009, Zimmermann, Grundler, Tai and Leymann 2007, Zimmermann et al. 2008). However, all presented approaches propose to document the usage of patterns during the architecting process, while this chapter focuses on using patterns in architecture decision recovery, where large parts of the original reasoning is not explicitly available any more.

5.3 Design of the experiment

For the design of the experiment, the guidelines by Kitchenham et al. (Kitchenham et al. 2002) and Wohlin et al. (Wohlin et al. 2012) were used. The former present general guidelines for software engineering experimentation and give some advice concerning the design, execution, analysis, and presentation of empirical studies without going into detail. The latter present the phases in more detail, discuss statistical tests and their suitability for different types of studies. In this experiment, Kitchenham et al.'s guidelines were primarily used in the planning phase of the experiment, while Wohlin

[1] In the remainder of this chapter, for simplicity, we will use the word pattern meaning software architectural pattern

et al.'s advice was used as a reference for the analysis and interpretation of the results. Jedlitschka's and Pfahl's reporting guidelines (Jedlitschka and Pfahl 2005) are used to describe the experiment in this chapter. The following subsections of the proposed template were left out, because they were not applicable, or the content was already presented in other sections: Inferences are discussed Section 5.6; impacts of the approach on time and quality are discussed in Section 5.5; interpretation and general limitations of the study are discussed in Section 5.6.2. The usage of this template introduces a certain level of redundancy, because a distinction between the design and the actual execution of the experiment is made. Some subsections of the execution phase are similar to corresponding subsections of the design phase.

5.3.1 Goal, hypotheses, parameters, and variables

The goal of the experiment is to find out, if architecture decision recovery that is based on systematic identification of patterns in the architecture leads to higher quality or quantity of recovered decisions compared to architecture decision recovery that is performed ad hoc and intuitively. Although systematic approaches for architecture decision recovery exist (e.g. (Jansen, Bosch and Avgeriou 2008)), practitioners in the industry still perform recovery in an ad-hoc, intuitive way.

The study goal led to the following null hypotheses and corresponding alternative hypotheses:

H_{01}: Focussing on identifying patterns in architecture decision recovery leads to lower or equal *quality* of recovered decisions compared to ad-hoc, intuitive recovery.

H_1: The *quality* of recovered decisions is higher when the recovery focuses on identifying patterns in the architecture, compared to ad-hoc, intuitive recovery.

H_{02}: Focussing on identifying patterns in architecture decision recovery leads to lower or equal *quantity* of recovered decisions compared to ad-hoc, intuitive recovery.

H_2: The *quantity* of recovered decisions is higher when the recovery focuses on identifying patterns in the architecture, compared to ad-hoc, intuitive recovery.

Dependent variables

Two dependent variables were observed during the experiment, as shown in Table 5.1: the quality and the quantity of recovered architecture decisions.

The following aspects are taken into consideration to measure the quality of the recovered decisions:

- Problem / Issue: The architectural design issue that is addressed by the decision.
- Decision: The outcome or solution imposed by the decision.
- Alternatives: Possible alternative solutions addressing the design issue.

- Arguments: A justification for the chosen decision instead of the alternatives.
- Requirements: Functional and non-functional requirements that are satisfied or affected by the decision.
- Related Decisions: Decisions that are related to or imposed by the current decision.

The quality of the recovered decisions was assessed by two independent experts in the field of software architecture, later also referred to as *analysts*, using a five point Likert-scale (see (Trochim 2001) for a description of Likert-scales) that ranges from one for *very poor* to five for *very high* quality. The ratings were left to their own experience and interpretation, but they were asked to take the aforementioned aspects of decision quality into consideration.

Quantity of architecture decisions is defined as the number of recovered architecture decisions. Decisions that both analysts concordantly rated as non-architectural would be excluded from the analysis. As it is hard to clearly estimate in how far a design decision concerns the architecture of the system, we provided some examples of architecture decisions and left further evaluation to the expertise of the analysts.

Description	*Scale Type*	*Unit*	*Range*
Quantity of recovered decisions	ratio	decisions	Positive natural numbers including zero.
Quality of recovered decisions	interval	n.a.	Five point Likert-scale. One for *very poor*, Five for *very high*.

Table 5.1: Dependent variables

Independent variables

The goal of the experiment was to discover the influence of patterns on the quality and quantity of decisions, obtained from architectural recovery. Therefore, two different treatments were defined for the participants. One group of participants was explicitly told to focus on identifying patterns in the architecture documentation; the participants in the other group did not get any specific advice, but they were allowed to perform this task as they would normally do it. The first group is referred to as *pattern group*, the latter as *control group*.

Table 5.2 shows other variables that could have an influence on the dependent variables. They relate to characteristics of the participants and mainly concern previous experience. In the design of the study, these variables were eliminated by defining blocking rules to balance the characteristics among the pattern group and the control group.

Description	*Scale Type*	*Unit*	*Range*
Group	nominal	n.a.	Possible values: *Pattern group*, *Control group*.
Affiliation	nominal	n.a.	Possible values: *university/academia*, *industry*, *other*
Programming experience	ordinal	years	4 classes: 0, 1-3, 3-7, > 8
Architecture experience	ordinal	years	4 classes: 0, 1-3, 3-7, > 8
Middleware experience	ordinal	years	4 classes: 0, 1-3, 3-7, > 8
Frequency of pattern usage in projects	ordinal	percent	4 classes: 0%, $< 25\%$, $> 25\%$, 100%
Number of well known patterns	ordinal	patterns	4 classes: < 5, 5-10, 11-20, > 20

Table 5.2: Independent variables

5.3.2 Experiment design

To test the hypotheses, we conducted two executions of a controlled experiment (Boehm et al. 2005) using exactly the same study design. The first execution took place at EuroPLoP 2009 (Hillside Europe e.V. 2009); the second execution took place during a software architecture workshop at the Fontys University of Applied Science in Venlo, the Netherlands, in April 2011.

Participants

The schedule of the EuroPLoP conference, where the first execution of the study took place, had reserved time slots for so called *focus groups* (FG). Attendees could propose topics in advance and publish them on the conference website. Multiple FGs were scheduled concurrently, so participants had to make a selection. We announced a focus group in advance and stated explicitly that we were planning to do an experiment on architecture recovery, based on an existing architecture documentation. The participation in the focus group was voluntary, but all participants who took part in the focus group also had to take part in the experiment. It was assumed that the primary motivation for taking part in the experiment was personal interest in architecture recovery. We expected to have 10 to 15 participants, based on experience from former focus groups. A background in at least one software-engineering discipline was presumed.

The second execution during the software architecture workshop in Venlo was an-

nounced as a practical session on architecture decision recovery. The participation in the workshop was free. Invitations were sent to alumni students from the hosting university of applied science, and colleagues from their companies. We assumed that the greatest part of the participants would have a significant industrial background and expected between 15 and 30 attendees.

The experiment design described in the following subsections was followed in the same way in both executions of the experiment.

Object

The basis for the architectural recovery was a five page document about the JBoss J2EE application server version 2.2.4, an excerpt of a research article on the JBoss architecture written by Jenny Liu from the University of Sydney in April 2002 (Liu, J. 2002).

JBoss, in the described version, is a free open source application server implementing the J2EE specification. The documentation does not explicitly mention the usage of any pattern, but hints exist in form of component names. The name *RequestBroker* for example hints at the usage of the Broker pattern (Buschmann et al. 1996).

The document describes the conceptual architecture and lists technologies and frameworks used in the implementation. Besides text, some box-and-line diagrams are used to illustrate components, and control- and data-flow in parts of the architecture. The participants received a print-out of the document.

The architecture of the used JBoss server is dominated by a microkernel, which was implemented using the Java Management Extension (JMX). The major JBoss services are encapsulated in so called MBeans, which are managed by an MBean server that is part of JMX. JMX itself has a layered architecture; the agent layer contains the MBean server. The bottom layer communicates directly with the Java virtual machine. Please refer to (Liu, J. 2002) for the detailed description of the architecture.

Blocking

To be able to explicitly analyze the influence of patterns in architecture recovery, we split the participants into two groups. One group was asked to identify and document architecture decisions related to patterns, whereas the other group did not get corresponding advice. The goal was to reduce the effect of independent variables that might influence the results of the analysis.

Because of the rather small sample size, we decided not to assign the participants to the groups randomly, but to balance the groups explicitly based on affiliation (university, industry, other), programming experience (0 years, 1-3 years, 3-7 years, 8 or more years), architecture experience (0 years, 1-3 years, 3-7 years, 8 or more years) and experience with object-oriented middleware (0 years, 1-3 years, 3-7 years, 8 or more years).

Instrumentation

Table 5.3 shows an overview over the instruments used in the three phases of the experiment. In the introduction phase, we asked all participants to fill in a questionnaire

Table 5.3: *Instrumentation overview*

Phase	Instrument	Purpose
Introduction	First questionnaire	Gather information needed for blocking
	Example decision	To explain the concept of architecture decisions
Experiment	Blank decision templates	Used by the participants to document the recovered decisions
	Pattern catalog	Provided to the pattern group as pattern reference
Wrap-up	Second questionnaire	Gather information needed for interpretation and validation of the results

prior to the recovery exercise, to gather information needed to perform the blocking (affiliation, programming experience, architecture experience and middleware experience). Unique random numbers were attached to the questionnaires to identify the participants throughout the experiment. They were also mapped to every recovered architecture decision.

In the same phase, we introduced the concept of architecture decisions to all attendees and presented one elaborate example on how to recover and document a decision, based on a small part of an architecture documentation. The example decision was handed out to all participants, so they could use it as a guideline during the experiment. The template used to document the decision was taken from Tyree and Akerman (Tyree and Akerman 2005).

In the next phase, the participants were asked to document the recovered decisions based on the same template. Therefore, we handed out as many blank templates on paper, as needed by the participants. Some fields in the template were optional, whereas *Problem/Issue*, *Decision*, *Arguments*, and *Related Requirements* were marked as mandatory fields. We encouraged the participants to provide as much information as possible regarding at least the mandatory fields.

Every member of the pattern group additionally received a printed copy of the most well-known architectural patterns (Avgeriou and Zdun 2005, Buschmann et al. 1996). Using the catalog to identify patterns was optional. It was assumed that many of the participants had knowledge about architecture patterns anyway. However, the catalog was provided to serve as a reference and as a reminder for the participants to focus on patterns. Any patterns or architectural styles were allowed that were used to solve an architectural problem. As described in Section 5.3.1, we left it up to the analysts to

judge, whether a recovered decision was architectural, or not.

An additional questionnaire, which we later also refer to as second questionnaire, was designed to gather further information from the participants after the experiment in the wrap-up phase. It contained questions concerning previous experience with software patterns and the usefulness of patterns during the recovery. Although this data is not needed for testing the hypotheses, it is useful for the interpretation and validation of the results. We asked these questions *after* the experiment for two reasons. First, because the questions regarding patterns could have influenced the participants of the control group (the non-pattern group) prior to the experiment; they could have guessed that patterns play an important role in the other group, and consciously or unconsciously also focus on patterns. Second, because we were interested in the way the participants actually performed the recovery.

Blinding

To eliminate subjective bias on the part of the participants and the experimenters, double-blinding was applied in the experiment. Although the participants had to realize that there are two different groups, they were not able to understand the purpose of the group division, the difference in treatments, and if they belong to the experimental group or the control group. To prevent the experimenters from being biased, the participants handed in the results using a participant number that would not allow to draw conclusions on their real identity. The participant numbers were assigned to them on the first questionnaire.

Because the experimenters necessarily know which participant number belongs to which group, the quality ratings of the results were done by two independent experts. We asked two people from our professional network to do the analysis. Table 5.4 summarizes the relevant experience of the analysts. The data was gathered using a web questionnaire.

The analysts did not get any specific information about the experiment. They were just asked to rate the quality of some documented decisions on a scale from one to five, as described in the variables section. Before handing the decisions out to the analysts, we pseudonymized them a second time by attaching a unique random number to every decision, and by internally mapping it to the participant number. That way, it was impossible for the analysts to find out which decision belongs to which participant, which decisions belong together, and which decisions belong to the pattern group. As mentioned earlier, the fact that there were two groups was not communicated to the analysts either.

Data collection procedure

After 30 minutes of introduction and grouping, the participants started with the recovery. The provided templates had to be used to document the recovered architecture decisions on paper. The participants of the groups were distributed over two separate

Table 5.4: *Characteristics of the analysts*

Characteristic	Analyst 1	Analyst 2
Working experience in the industry	9 years	33 years
Experience in the field of software architecture	9 years	12 years
Experience with object-oriented middleware like J2EE	5 years	3 years
Involved in making architecture decisions (5-point Likert-scale from *very frequently* to *very rarely*)	very frequently	frequently
Involved in documenting architecture decisions (5-point Likert-scale from *very frequently* to *very rarely*)	very frequently	frequently
Involved in the analysis of architecture decisions (5-point Likert-scale from *very frequently* to *very rarely*)	very frequently	frequently

rooms according to the group membership. Two experimenters were present in each room to answer questions related to the instructions and to take care that participants did not communicate with each other. Once the session was completed, the documented decisions were collected by the experimenters. Finally, a wrap up session was planned to collect comments on the experiment and to fill in the questionnaires about pattern experience and pattern usage mentioned in Section 5.3.2. Including a 30 minute break, the experiment lasted three hours.

5.4 Execution

5.4.1 Sample and preparation

As described in the design section, the experiment was announced as a focus group during EuroPLoP 2009 and as a practical session on architecture decision recovery at the workshop in Venlo.

At EuroPLoP, twelve people were willing to take part in the experiment, from which one had to be rejected because of a lack of software engineering experience. Twenty-two people took part in the practical session in Venlo, from which none had to be rejected.

All participants filled in the first questionnaire and were afterwards assigned to either the pattern group or the control group. The blocking procedure went as expected, according to the experimental design.

Figure 5.4.1 shows previous experience and affiliation of the participants, as assigned to the pattern group and the control group. The figures accumulate the data from all participants from the two executions of the experiment.

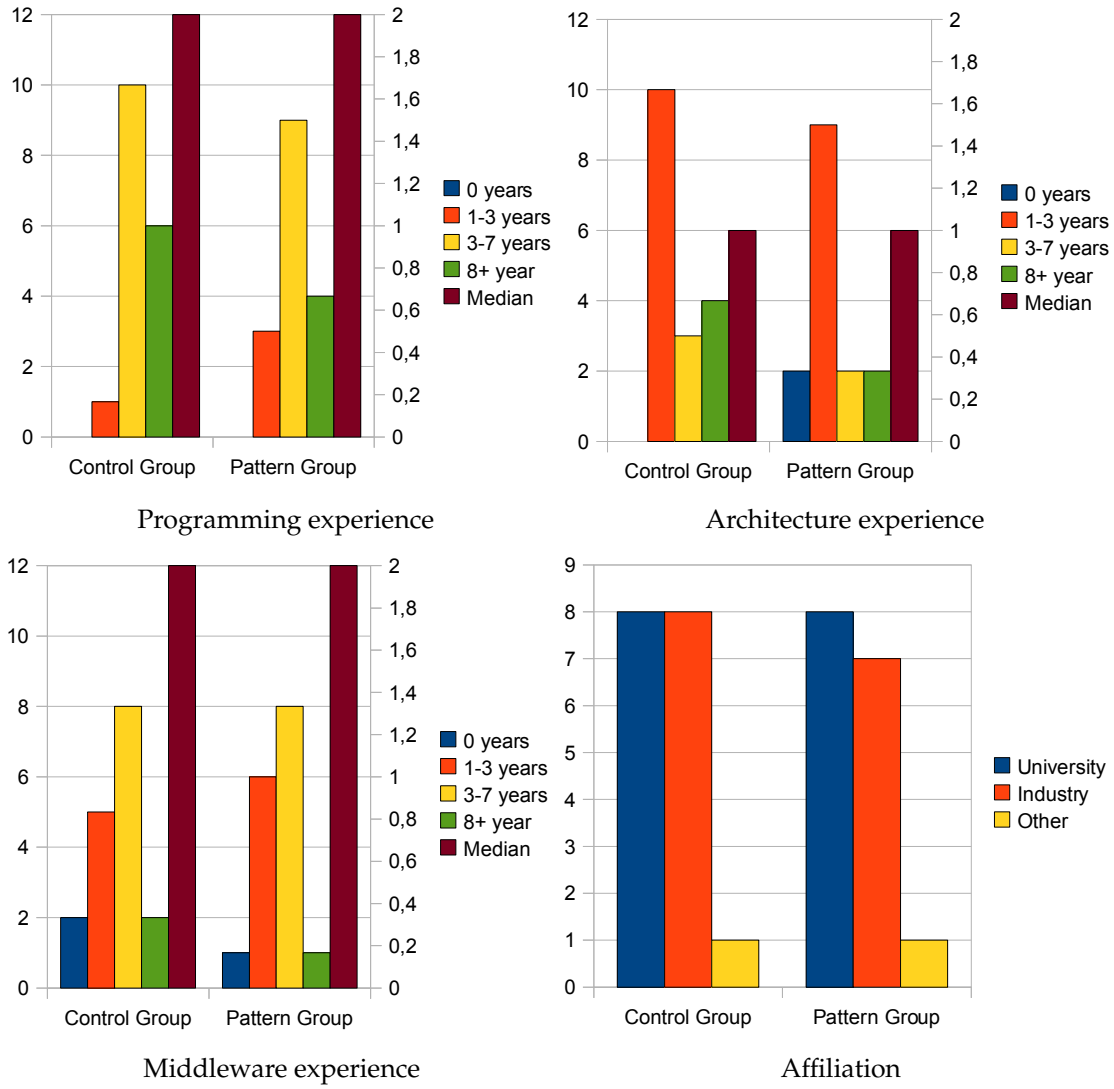


Figure 5.1: *Distribution of participants*

Subfigures (a) to (c) show the previous experience of the participants concerning programming, architecture and middleware. Additionally to the total numbers of participants in each class, median values are shown for each group. The medians are aligned to the right vertical axis, whereas all other values are aligned to the left vertical axis. Medians were calculated as follows. First, each of the year-intervals was assigned to a single value in an ordinal scale ranging from zero to three. '0 years' was assigned to the value 0, '1-3 years' was assigned to 1 and so on. Then medians were calculated based on the ordinal scale. In total, the median programming-, architecture- and middleware experience for both groups is two, which means that the participants in the groups were balanced concerning their previous knowledge.

At the same time, the participants from both groups were introduced to the concept of architecture decisions using the prepared recovered decision. The introduction took approximately 15 minutes.

5.4.2 Data collection performed

The data collection at the EuroPLoP execution was performed as planned in the design. No participants dropped out and no deviations from the study design occurred.

In the workshop execution in Venlo, one of the participants from the control group did not hand in the recovered decisions after the experiment. Consequently, his data could not be taken into consideration. Other than that, everything went as planned.

5.4.3 Validity procedure

The experiment took place in a controlled environment. The participants were assigned to two different rooms according to their group (pattern, or control group). At least one experimenter was present in each room during the whole experiment time to assure that participants did not use forbidden material and did not talk to each other. After the experiment, all documented decisions were collected by the experimenters before any of the participants left the room. There were no situations in which participants behaved unexpectedly.

5.5 Analysis

5.5.1 Descriptive statistics

We use descriptive statistics to visualize the collected data as a first step in the analysis. The first two subsections are related to the hypotheses tests: Section 5.5.1 presents an analysis of the quality of documented decisions. Section 5.5.1 concerns the quantity of decisions. The last subsection presents an analysis of the data gathered in the second questionnaire, in which the participants were asked about their previous experience and the usefulness of patterns during architecture recovery. The results are compared to those of the analysis of the quality and quantity of the recovered decisions.

Quality of recovered decisions

As explained in the design section, the quality of knowledge in every recovered decision was rated by two independent experts using a five point Likert-scale, ranging from one for *very poor quality* to five for *very high quality*.

The level of scaling (e.g. nominal, ordinal, interval, ratio) for Likert-scales is hard to determine. It is common sense that Likert-scales are at least ordinal in nature (Goldstein and Hersen 2000). For the quality ratings in this experiment, this is given. A decision that was rated with five has a higher quality than a decision rated with four. However, to be able to use parametric statistical tests like the t-test, at least an interval scale (Stevens 1946) character of the scale must be assumed. This is the case, if equal distances between the points on the scale can be assumed, e.g. the difference between the ratings five and four would be the same as the distance between ratings two and three. In our specific

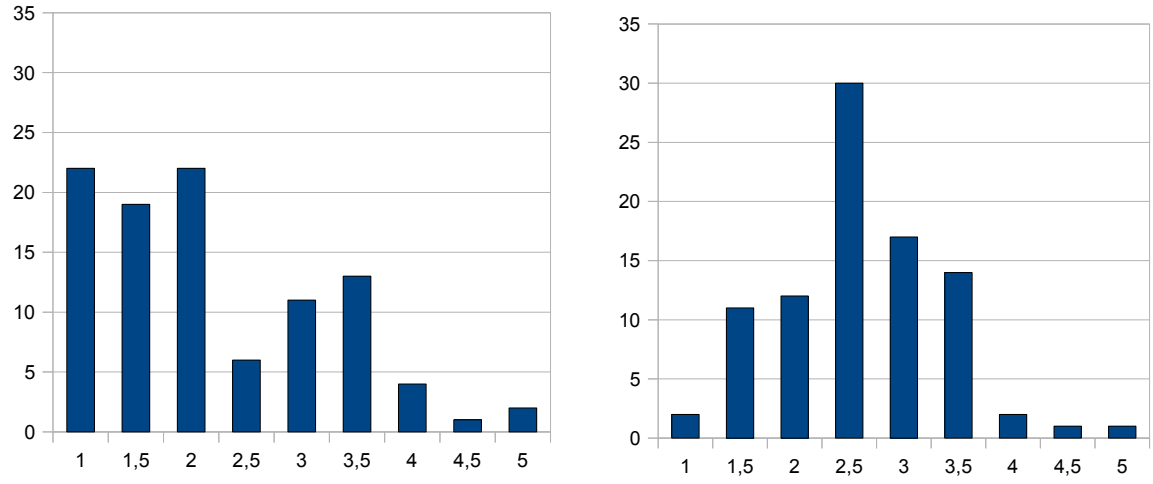


Figure 5.2: *Frequency: quality of decisions*

case we assume that this holds true.

In the analysis of the experiment, a t-test for independent variables (O'Gorman 2004) is used to calculate the significance of the found results. Levene's significance test is used to find out whether equal variances of the quality ratings can be assumed.

Based on the data in Table A.1, the following descriptive statistics apply for the quality of knowledge. Figure 5.2 shows the frequency of rated quality for the decisions recovered by the pattern group and the control group. From the figure, we see that the quality of decisions in the pattern group seems to be higher than the quality of decisions in the control group. Moreover, the most frequent quality ratings in the pattern group are 2.5 (33.3%) and 3 (18.9%), compared to 1 (22.0%) and 2 (22.0%) in the control group.

Table 5.5: *Additional descriptive statistics*

	Control Group	Pattern Group
N	100	90
Mean	2.185	2.611
Std Dev	1.032	.752
Variance	1.064	.566
Median	2.000	2.500

As argued before, we interpret the Likert-scale as an interval scale; so the mean, standard deviation, variance, and range apply as measures. Additionally, the median value is calculated, which would also be applicable for ordinal scales. Table 5.5 shows a comparison between the statistics for the control group and the pattern group. Besides the fact that the average quality of decisions in the pattern group is higher than in the control group, the variance in the control group is much higher than the variance in the pattern group. This means that the dispersion of quality ratings is higher in the control group.

Quantity of recovered decisions

Table 5.6: *Descriptive analysis quantity of decisions*

	Control Group	Pattern Group
N	16	15
Mean	6.25	6.0
Std Dev	4.386	3.359
Variance	19.267	11.286

The quantity of recovered decisions is measured counting all architecture decisions that were not excluded as being non-architectural by both analysts. One of the analysts excluded some decisions as being non-architectural, whereas the other analysts did not exclude decisions at all. Because there was no mutual agreement on any of the cases to be excluded, we included all decisions in the quantitative analysis.

The number of participants in the two groups is shown in Table 5.6, together with the mean values, the standard deviations, and the variances for the respective group. The mean quantity of decisions, which is measured in terms of the number of recovered architecture decisions, is slightly higher in the control group than in the pattern group. The standard deviation in the control group is considerably higher than in the pattern group.

Analysis of second questionnaire

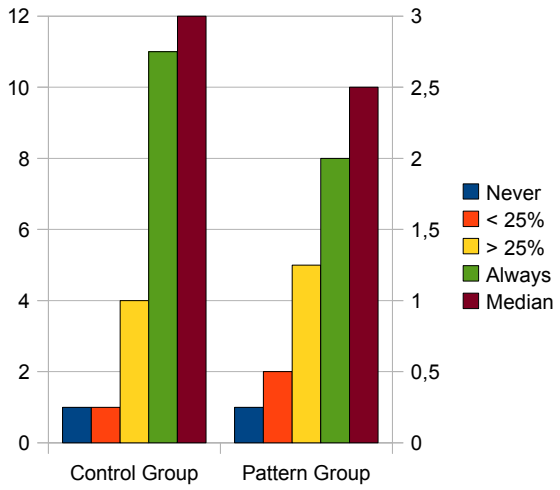
In this section, the results from the second questionnaire, which was filled in after the experiment took place, are presented.

Figure 5.3 (a) shows the frequencies of answers to the question: *How often do you apply software patterns in your software projects?* Possible answers were *Never*, *In less than 25% of the projects*, *In at least 25% of the projects*, *In every project*. Additionally to the frequency of answers, median values are shown. They were determined by assigning each of the answers to a single value in an ordinal scale ranging from zero to three and calculating the medians based on these numbers. While all other values are aligned to the left vertical axis, the medians are aligned to the right vertical axis. The figures show that the median for the control group is higher than for the pattern group. This means that the participants had more pattern and recovery experience than the participants in the pattern group.

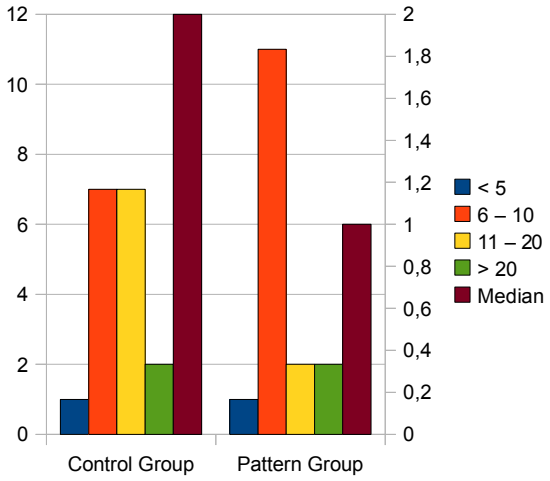
The frequencies of answers to the question: *How many software patterns do you know well?* are shown in Figure 5.3 (b). Possible answers were *Less than 5 patterns*, *Six to ten patterns*, *Eleven to twenty patterns* and *More than 20 patterns*.

The participants were also asked whether they did architectural recovery before. The results are shown in Figure 5.3 (c).

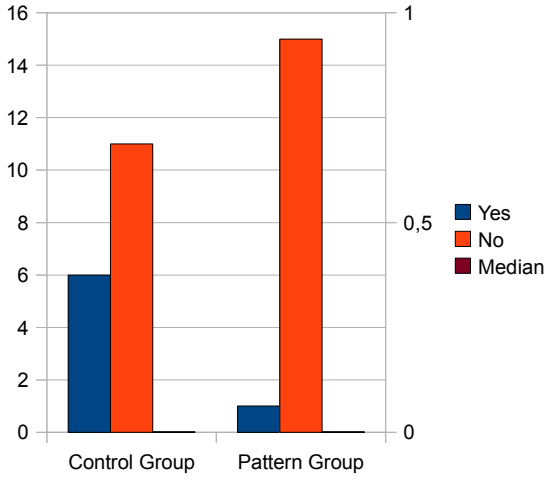
The next two questions from the second questionnaire concerned the usage and helpfulness of patterns and show a small delta between the groups. First, the participants were asked to rate the helpfulness of patterns during architecture recovery on a



(a) How often do you apply patterns in projects?



(b) How many patterns do you know well?



(c) Have you done architecture recovery before?

Figure 5.3: *Pattern and architecture recovery experience*

scale from one for *not helpful* to five for *very helpful*. The median in both groups is four. These results are subjective in nature, as they express opinions. However, it shows that the members of both groups generally consider patterns as very useful in architecture recovery.

In the next question, the participants were asked to estimate how extensively they used patterns during the recovery. Possible answers ranged from one for *almost never* to five for *very often*. The median answer in the pattern group is three, the median answer in the control group is two.

We looked into the types of the recovered decisions to see if the subjective estimations of the participants reflect the reality or not. In particular, the decisions were classified into pattern-related decisions, if the name of a pattern is literally mentioned in the documented decisions, and others, i.e. non pattern-related decisions. The results of this analysis can be found in Table A.1.

The average number of pattern-related decisions per participant in the pattern group is 4.6 compared to 1.88 in the control group. The average number of other decisions per participant in the pattern group is 2.07 compared to 4.38 in the control group. The ratio of the pattern-related type to the other type is 2.23 in the pattern group compared to 0.43 in the control group. This shows that the members of the pattern group clearly focused more on patterns than the members of the control group. Independently from the group, in which decisions were taken, the median quality of pattern-related decisions is 2.5, the median quality for other decisions is 2. This analysis of decision types has two results. It verifies that the pattern group focused on identifying pattern-related decisions and it shows that the difference in quality of decisions presented above can be ascribed to the focus on patterns.

Finally, we asked the participants to briefly describe how the recovery was performed. This was primarily done to confirm that the pattern group followed a pattern-based approach and to find out if the control group used any other systematic way to identify and describe decisions. Although the amount of qualitative data for this question was low (roughly one sentence per participant), we use the constant comparative method, as originally described by Glaser and Strauss (Glaser and Strauss 1967) to systematize the analysis of the answers. Therefore, we grouped (partial) answers to the question how the recovery was performed into categories. Each answer was compared to the previously coded answers in the same and other categories to gain a better understanding of the decision recovery process they describe. Finally, the categories elicited from the control group were compared to the categories from the pattern group.

The results imply that the control group followed an intuitive approach, which was mainly driven by personal experience. Four participants answered that they searched for buzzwords that would remind them of a familiar technical solution. Three respondents stated that they read the textual descriptions in the architecture document to mine decisions; two analyzed the given UML diagrams. Three participants from the control group explicitly answered that they searched for patterns in the architecture. The other answers were not assigned to a specific category. However, one of these answers extremely represents the impression we gained during the analysis of the answers of the control group: “It looks like decision -¿ it is decision”.

The answers of the pattern group reflect the focus on patterns. Thirteen out of 18 answers explicitly described an approach that centers on patterns. Six participants answered that they searched the UML diagrams for potential pattern participants. Four respondents identified candidate pattern decisions in the architecture documentation and then read up on the pattern in the pattern catalog, before they documented the decision using the given template.

5.5.2 Data set reduction

Outliers are potential candidates for dataset reduction, i.e. data points that are either much higher, or much lower than other data points. To find potential outliers, we calculated the average quality of decisions for each participant. Figure 5.5.2 shows bar

charts for every member of the control and the pattern groups for both executions of the experiment. The first two figures represent the participants at EuroPLoP 2009, the latter two represent the participants from the software architecture workshop in 2011. The numbers in the legends are the participant numbers.

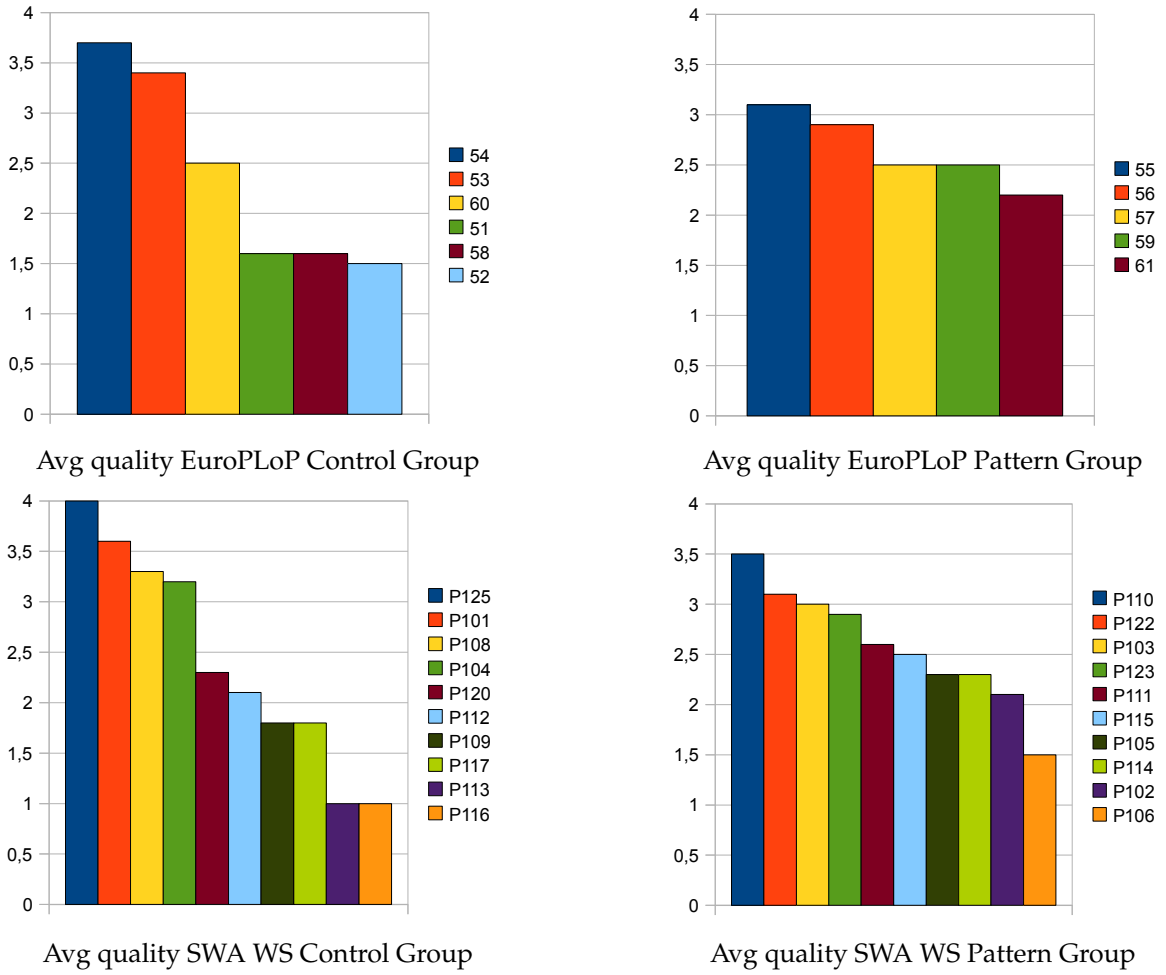


Figure 5.4: *Average quality per participant*

Two participants from the control group at EuroPLoP (Figure 5.4 reached a significantly higher quality than the other members of this group. A closer analysis showed that most of their decisions concerned patterns. Three out of five decisions from participant 54 were pattern decisions; six out of eight decisions from participant 53 were pattern decisions. This could lead to the conclusion that the high decision quality of these two participants results from the focus on patterns. However, their decisions were not excluded as outliers, because the difference to the other participants was not strong enough. Additionally, excluding the data points would have introduced a potential vulnerability of the study results.

5.5.3 Hypothesis testing

The two hypotheses, regarding higher quality and quantity of recovered decisions when architecture recovery is focused on identifying patterns, are evaluated using t-tests.

Quality of decisions

Table 5.7: *Independent t-test for quality of decisions*

Factor	Mean diff.	t-value	p-value
control group vs. pattern group	-0.4261	-3.222	0.001

The results from the t-test (unpaired, two-tailed) are shown in Table 5.7. It provides strong evidence that H_{01} can be rejected. There is a noticeable difference in the quality of the recovered decisions between the pattern group and the control group. The p-value is very low, so the results are highly significant. Even if the classification of the used Likert-scale for the quality ratings of the decisions as interval scale could not be accepted, the descriptive statistics would still strongly support the result of the t-test, as the median value and the frequency of measured quality both support a result in favor of the pattern group.

Quantity of decisions

Table 5.8: *T-Test independent samples test quantity of decisions*

Factor	Mean diff.	t-value	p-value
pattern group vs. control group	-0.250	-0.177	0.861

Hypothesis H_{02} was also evaluated with a t-test (unpaired, two-tailed). The results are shown in Table 5.8. Although slight differences in terms of the mean values can be observed, we are unable to show that this result is significant. An exclusion of the cases that were rated as non-architectural by one of the analysts would not have had an impact on this result.

5.6 Interpretation

5.6.1 Evaluation of results and implications

Quality of decisions

Hypotheses H_{01} and H_1 concern the quality of recovered decisions. As pointed out in Section 5.5, we are able to provide strong evidence that the null-hypothesis H_{01} can be rejected.

Thus, the quality of decisions gained during architecture recovery is higher if the recovery focuses on identifying applied patterns. Additionally to the generally higher quality, the variance in the pattern group is much lower than in the pattern group.

We interpret our findings as follows. Patterns provide rich information about their problem- and solution spaces as well as reasoning for applying them in a system. They contain a great part of the architectural knowledge that is relevant for the system, in which they were applied. If a pattern was identified during the recovery process, then the pattern documentation or the personal knowledge about the pattern helps to recover the intent of the original architect, who decided to apply it. Of course, it still takes some effort to identify the pattern and customize the pattern's documented knowledge for the system at hand; but a large part of that high-quality knowledge is reused, not invented. The fact that the variance in the pattern group is relatively low shows that patterns help to reduce the dependency on individual abilities of the person doing the recovery. A certain quality level can be achieved even by people who do not have a strong background in architecture recovery. Consequently, the higher variance in the control group might stem from the different abilities of the participants.

Quantity of decisions

Hypotheses H_{02} and H_2 concern the quantity of recovered decisions. The results do not provide evidence to confirm or reject the null-hypothesis H_{02}. We are unable to show that the focus on patterns in architecture recovery has a significant effect on the number of recovered decisions. This result is surprising to us. As described in the introductory section, we assumed that the quantity of recovered decisions would be higher in the pattern group. The results might stem from the fact that the participants in the pattern group took more time to document every single decision than the participants in the control group and thus had less time left to identify decisions. They also needed time to study the pattern catalog. This effect could possibly be eliminated by adjusting the study design. We will discuss this in Section 5.6.3.

The variance of the quantity was much higher in the control group than in the pattern group (19.267 compared to 11.286). This is another indicator for the lower dependency on the recoverer's personal skills and abilities as already discussed for the quality of decisions.

5.6.2 Limitations of the study

Several levels of validity have to be considered in this experiment. We consider the classification scheme for validity in experiments by Cook and Campbell (Cook and Campbell 1979). Internal validity concerns the cause effect relationship between the treatment and the dependent variables measured in an experiment. External validity focuses on the generalizability of the results for a larger population. Conclusion validity focuses on the relationship between treatment and outcome and on the ability to draw conclusions from this relationship. Finally, construct validity is about the suitabil-

ity of the study design for the theory behind the experiment. All threats to validity are categorized according to this classification.

Internal validity

- The object in the experiment was a documentation of an object-oriented middleware. In this particular case, the JBoss application server, many architectural patterns were implicitly and explicitly applied in the system, which might lead to the conclusion that the pattern group had advantages compared to the control group. This, however, does not seem to be the case. Both groups could have identified the architecture decisions behind the applied patterns. Also many other architecture decisions were made by the original architects that do not concern patterns, e.g. the choice of used frameworks or programming libraries. Finally, although many patterns were applied in the JBoss server, our results do not confirm that a focus on patterns leads to higher quantity of decisions. Thus, the fact that the JBoss design contains a lot patterns did not have an effect in our study.

  Another potential threat related to the choice of JBoss as object of the study is the fact that many J2EE patterns exists. The former SUN catalog of J2EE patterns is one source of such patterns (Oracle Corporation 2002). However, the J2EE patterns support the creation of applications that conform to the J2EE specification set. To the best of our knowledge, no pattern catalog or pattern language exists that is specific to developing J2EE servers. In this case, the application analyzed by the participants was a J2EE server, not a J2EE application. Thus, we do not consider this a threat to validity.

  We conclude that the choice of the object studied in this experiment is not a threat to the internal validity of the results.

- The outcome of the experiment could have been different for systems, in which fewer or no patterns were applied by the designers. Normally, it is more difficult to identify pattern decisions in systems, in which not many patterns have been applied. However, a study of pattern usage, conducted by Harrison and Avgeriou (Harrison and Avgeriou 2008), showed that most systems have at least two architecture patterns, some have as many as eight. Furthermore, besides architecture patterns, the pattern community has assembled a vast body of pattern knowledge for virtually all software domains. Thus, several patterns can be potentially found in any system. Moreover, even if patterns are not consciously used by designers, they can still be applied unconsciously, as designers tend to reach common solutions. It is unlikely that all architecture decisions in a system are pattern related, but even in cases where only a few patterns were used, the decisions can be an important entry point for the recovery of the remaining decisions, because decisions are usually interrelated. The threat, however, cannot be mitigated completely.

  In a few rare cases, so many patterns could have been applied in an architecture that individual patterns are hard to identify in the design. This, however, is a

theoretical problem that is not very likely to be observed in reality. We do no consider it a threat to validity.

- Typically, there is a variation in human performance that might influence the results of the experiments. This can distort the results, because then the performance would not arise from the difference in treatments. We tried to minimize this factor by balancing the two groups concerning the relevant previous experience of the participants. The groups were well balanced in all categories, namely programming experience, middleware experience, architecture experience and recovery experience. Thus, this factor is not seen as a threat to validity.

- The control group could theoretically have imitated the behavior of the pattern group. In this particular experiment, the two groups performed in two different rooms at the same time. The instructions that concerned the difference in treatments were given to the participants after they moved into these rooms. That way, there was no chance for the control group to consciously or unconsciously imitate the behavior of the pattern group.

- The raters could have unconsciously ranked the pattern decisions higher than other decisions, because patterns contain professionally edited material that is succinct and easy to comprehend. The data gathered in both executions, however, shows that the participants used the patterns to interpret the architectural solutions found in the JBoss architecture and documented the decisions using their own words, adapting the pattern information in the context of the JBoss system. Therefore, decisions by and large, were not documented by copying or reusing the text from the pattern catalogs.

External validity

- The subject population in the experiment might not be representative for a larger population. In this case, the subjects (participants) of the first execution of the experiment were participants of the EuroPLoP conference. They all have an academic or industrial background in several software engineering disciplines and a strong interest in patterns. The second execution at the software architecture workshop in Venlo was conducted mainly with industrial practitioners from different domains. Our results imply that the affiliation (industry or academia) does not have an influence on the external validity of the results. No correlation between the affiliation and the quality of recovered decision could be found. Additionally, each of the two executions analyzed in isolation would have lead to the same conclusions, namely that a focus on patterns leads to higher quality, but not to higher quantity. Therefore, we conclude that the pattern background of the EuroPLoP participants does not distort the study results.

- The instrumentation and object in the experiment might have been unrealistic or

old-fashioned. In this case, the architecture recovery was based on a printed architecture documentation. Usually different tools would be used

to support architecture recovery. Code analyzers, reverse engineering tools and dependency analysis tools are some examples. These tools are primarily used to recover the design of a software system. In this experiment, for practical reasons, the design of the software was readily provided in a printed document. The focus was on architecture decision recovery, not on architecture design recovery. We assume that the measured effect of a pattern focus during architecture decision recovery is independent from the way, in which the design was recovered.

Another theoretical thread to validity is that the problem in the analysis might be unrealistic and too simple to allow generalization. This was not the case here. The object used is an excerpt from a real documentation of the JBoss server that was not created for the purpose of this experiment.

- Finally, the experimenters could have biased the measurements of the independent variables. We mitigated this risk by assigning the quality ratings of the decisions to two independent experts that had no knowledge about the goals of the experiment. Additionally, by using pseudonymization, the analysts had no chance to guess which decisions belonged to which group. They could not even have found out which decisions belonged together, i.e. were documented by the same participant.

Conclusion validity

- As discussed in the design section, there is a potential threat to validity resulting from the interpretation of the Likert-scale, which was used to rate the quality of architecture decisions, as an interval scale. Some of the statistical tests used to analyze the results (mean, variance, standard deviation and t-test) would not have been valid for nominal scale types. We argue that in this particular situation the ratings of the Likert-scale are metrically scaled, and thus have the character of an interval scale. This means, for instance, that the quality rating four is actually two-times higher than the quality rating two. Because this interpretation remains critical, we also calculated the median for the quality ratings, which would also be applicable for nominal scales.
- Another potential threat to validity is the subjectivity of the scale used to rate the quality. We tried to mitigate this risk by asking two independent experts in the field of software architecture to rate the quality of every recovered decision. In the analysis, we took the arithmetic average of the two ratings per decision as a basis. However, the null-hypothesis would also have been rejected for the results of both analysts individually. Additionally, from the fact that our result has a very high significance, we conclude that this potential threat is mitigated.

Construct validity

- The fact that only one object; the JBoss documentation; was used in the experiment, introduces the risk that the cause construct is underrepresented. Theoretically, the results could look different if multiple architecture documentations would be used for the recovery. We assume that the used system and its documentation are representative for large and medium-size object-oriented systems. The threat, however, cannot totally be ignored.

- Another potential threat to validity is the number of measures used to evaluate the quality of recovered decisions. In our case we only used one variable to measure the quality of the recovered decisions. This does not allow cross-checking the results with different measures.

5.6.3 Lessons learned

The analysis of the quantity of decisions showed that, on average, the control group recovered more decisions than the pattern group. We already presumed that one of the reasons for this outcome might be that the participants in the pattern group took more time to document every single decision than the participants in the control group and therefore had less time left to identify decisions. Besides, they took time to study the pattern material. The latter was particularly the case during the workshop in Venlo. The participants at EuroPLoP had presumably more knowledge about patterns and consequently took less time to study the pattern material.

One way of eliminating this effect would have been to assign more time to the pattern group than the control group. But, as the additional time needed to document decisions can hardly be estimated or even predicted, it would have been hard to define an adequate period of time to add it to the pattern group's experiment run-time. Additionally, a potential threat to validity would have been introduced.

Another possible improvement of the study design concerns the data collection. In this experiment, we could not make use of computers or other electronic devices to collect data. This was a handicap during the analysis. Gathering data electronically using online surveys and electronic forms would have eased the analysis. However, participants felt comfortable with the architecture documentation on paper, because the paper form allowed them to take notes.

5.7 Conclusions and future work

In this chapter, we describe the results of a controlled experiment that was conducted to find out if patterns are beneficial for architecture decision recovery. Two aspects were specifically taken into consideration: the quality and the quantity of recovered decisions. The evaluation of the experiment shows that a focus on patterns leads to significantly higher and stable quality of decisions, compared to intuitive recovery, which

leads to a lower quality with higher variance. We are unable to show that the quantity of recovered decisions is also positively affected.

In the future, we plan to replicate the experiment with different types of software systems from other application domains, which are less pattern-intensive than the object used in this study.

Another direction for future work is to find out if besides patterns, there are other forms of generic architectural knowledge that can be beneficial in architecture decision recovery. In the context of a research project, we developed a publicly available online repository for patterns and technologies (University of Groningen, Software Engineering and Architecture Group 2012b). The basis for the repository is a common meta model for patterns and technologies that allows to relate patterns, pattern variants and software technologies. We plan to use the tool for a follow up experiment, in which we allow the treatment group to use all kinds of generic architectural knowledge, instead of focussing on patterns.

5.8 Acknowledgements

We would like to thank Anton Jansen and Chuck Allison for analyzing the results of the study. We also thank the members of the EuroPLoP 2009 focus group and the participants of the software architecture workshop in Venlo 2011 for taking part in the experiment.

Based on: U. van Heesch, P. Avgeriou, and R. Hilliard – *"A documentation framework for architecture decisions"*, Journal of Systems and Software, 85(4):795- 820, 2012.

Chapter 6
A framework for architecture decisions

Abstract

In this chapter, we introduce a framework for architecture decisions. This framework consists of four viewpoint definitions using the conventions of ISO/IEC/IEEE 42010, the new international standard for the description of system and software architectures. The four viewpoints, a decision detail viewpoint, a decision relationship viewpoint, a decision chronology viewpoint, and a decision stakeholder involvement viewpoint, satisfy several stakeholder concerns related to architecture decision management.

With the exception of the decision stakeholder involvement viewpoint, the framework was evaluated in an industrial case study. The results are promising, as they show that decision views can be created with reasonable effort, while satisfying many of the stakeholder concerns in decision documentation.

6.1 Introduction

With the growing complexity and size of software-intensive systems, software architecture has become increasingly important. While architecture is traditionally understood as the design of the system itself, manifested mainly in design elements and their form, Perry and Wolf recognized the importance of (design-)rationale as an integral part of the software architecture. They defined software architecture as follows:

$$Software\ Architecture = \{Elements, Forms, Rationale\}$$ (Perry and Wolf 1992)

Kruchten adopted this definition of software architecture as a starting point for the 4+1 View Model framework (Kruchten 1995). In this framework, each of the five views addresses various stakeholder concerns and determines the organization of a set of architectural elements, the forms and patterns used, and the rationale behind those architectural choices. This concept of documenting software architecture as a set of views that correspond to viewpoint (VP) definitions and address stakeholder concerns was adopted and generalized in IEEE Std 1471:2000 (IEEE 2000), and further elaborated by the architecture community (e.g., (Clements et al. 2010, Rozanski and Woods 2005)). However, as in Kruchten's 4+1, the importance of documenting decisions and their rationale along with the selected architectural concepts was only mentioned, but little guidance was offered on how to document decisions.

Bosch emphasized the importance of documenting architecture as a set of architecture decisions (ADs) (Bosch 2004). In contrast to the aforementioned approaches, design decisions as an explicit part of the software architecture description provide insight into the reasoning process and record the rationale behind design decisions. The concept of architecture decisions has been incorporated into ISO/IEC/IEEE 42010 (ISO/IEC/IEEE 2011), which is the international revision of IEEE Std 1471:2000 (IEEE 2000).

Today, the perspective of looking at software architecture in terms of a set of architecture decisions is widely recognized. Authors have proposed templates for the information that is important to capture about decisions (e.g., (Jansen and Bosch 2005, Tyree and Akerman 2005)), and various models and tools to capture and manage architecture decisions have been proposed (Tang, Avgeriou, Jansen, Capilla and Ali Babar 2010). Several approaches incorporate the documentation of architecture decisions in architecture practice and subsequently capture and organize architecture decisions to address various concerns, such as traceability and architectural conformance (Babar et al. 2009).

There are currently three main approaches to documenting architecture decisions: *decision templates*, *decision models*, and *annotations*. We argue that all three approaches satisfy some decision-related concerns, but none of them succeeds in satisfying all concerns. Shortcomings of architecture decision documentation approaches are not surprising: as with traditional architecture views, there is not a single way of documenting architecture decisions that frames all concerns of all stakeholders in an adequate and useful manner. We suggest that multiple dedicated viewpoints should be defined that focus on framing specific decision-related concerns.

In this chapter, we propose a framework consisting of four viewpoints for architecture decisions: A decision detail viewpoint, a decision relationship viewpoint, a decision chronology viewpoint, and a decision stakeholder involvement viewpoint. Each viewpoint is dedicated to framing specific decision-related concerns. At the same time, each viewpoint is integrated with the other viewpoints through a common metamodel to offer a more complete picture of decisions and their rationale. The framework proposed here is useful "out of the box", but it can also serve as a basis for customization or extension, by adding new decision-related viewpoints. One extension of the framework, the decision-forces viewpoint, is presented in Chapter 7. This viewpoint also builds upon the current framework metamodel. Apart from the decision stakeholder involvement viewpoint, all viewpoints were validated in an industrial case study with very promising results.

The rest of this chapter is organized as follows. Section 6.2 presents stakeholder concerns related to architecture decisions. In Section 6.3, we briefly outline the proposed viewpoints, including an example view for each of the viewpoints[1]. In Section 6.4, we report on an industrial case study, which was conducted to validate the viewpoints. Section 6.5 summarizes related work and Section 6.6 presents our conclusions and ideas for future work.

[1]The whole framework in terms of a unified metamodel, the complete viewpoint definitions, and the correspondences between those viewpoints, are specified in Appendix B.3.

6.2 Concerns related to architecture decisions

Architecture decisions should be documented to complement architectural design with rationale. Yet, how to capture decisions is still subject to discussion. This is mainly because there is no consensus on which stakeholder concerns must be addressed by a decision documentation approach. A *concern*, as used here, is any interest in a system

Table 6.1: *Concerns for architecture decision documentation*

Code	Concern
C1	What decisions have been made?
C2	What is the current set of relevant decisions?
C3	What is the rationale for decision D?
C4	What concerns C_i does decision D pertain to?
C5	Which requirements impacted/influenced each decision?
C6	What decisions D_k are influenced by requirement R?
C7	Which requirements R_l have conflicting influences on decision D?
C8	What decisions are required by decision D (including unmade decisions)?
C9	What decisions conflict with decision D?
C10	What decisions are dependent on decision D?
C11	What decisions are related to decision D?
C12	What decisions influence decision D, or architecture element E?
C13	What decisions are impacted by a change?
C14	What decisions would be impacted when integrating a set of decisions S?
C15	How to apply a set of decisions from a different project in the target architecture?
C16	Which stakeholders are affected by decision D?
C17	What decisions affect stakeholder S?
C18	Which stakeholders were involved in decision D?
C19	What decisions are influenced by stakeholder S?
C20	What is the ordering of decisions made?
C21	What decisions have changed since time T, or milestone M?
C22	What decisions became obsolete after change CH?
C23	What decisions D or decision sub-graphs SG can be reused in other projects?

on the part of its stakeholders. Each concern poses a question or issue that the architecture description, in this case the architecture decision documentation, should be able to answer.

In recent years, many use cases for architectural knowledge management have been published in the literature. We argue that decisions are one type of architectural knowledge. Therefore, we have analyzed three recent publications containing architectural

knowledge management (AKM) use cases (Liang et al. 2009, Kruchten et al. 2006, Jansen et al. 2007) to identify and derive concerns for architecture decision documentation. Table 6.1 shows the resulting concerns[2]. The concerns were functionally grouped and, where possible, ordered according to the authors' estimation of their importance. The actual importance of the concerns, however, often depends on the specific needs of the concrete stakeholder.

The analysis procedure, as well as a complete table with the analyzed use cases, the derived concerns, and the activities performed to derive the respective concerns, can be found in Appendix B.1.

Table 6.2: *Architecture decision concerns related to typical stakeholders*

Stakeholders	Concerns
Architects	C1, C2, C3, C4, C5, C6, C7, C8, C9, C10 ,C11, C12, C13, C14, C15, C16, C17, C18, C19, C20, C21, C22, C23
Reviewers	C1, C2, C3, C4, C5, C6, C7, C8, C9, C10, C11, C12, C16, C18, C20, C21
Managers	C17, C18, C19
Customers	C3, C6, C7
Requirements Engineers	C6, C7
New project members	C3, C20
Domain experts	C23

Next, the authors assigned the concerns to typical stakeholders. Table 6.2 shows the results. Most concerns were assigned to architects and reviewers, because these stakeholders are frequently using architecture documentation in their daily work. The assignment of the concerns took place based on typical tasks that the stakeholders perform in software projects. It could be argued that requirements engineers or managers, for instance, could also be interested in dependencies between decisions or the impact of a change in the architecture. However, we decided to limit ourselves to the most characteristic concerns for the respective stakeholders.

The concerns were taken as a basis for the development of the decision viewpoints, which will be introduced in the following section. Each of the viewpoint definitions was driven by the typical stakeholders and concerns it frames. With the exception of the concerns that were exclusively assigned to the stakeholder involvement viewpoint (C16, C17, and C19), all concerns for the viewpoints were validated as part of the case study presented in Section 6.4.

[2]A decision sub-graph, as used in concern C23, is a subset of a bigger set of interrelated decisions

6.3 A framework for architecture decisions

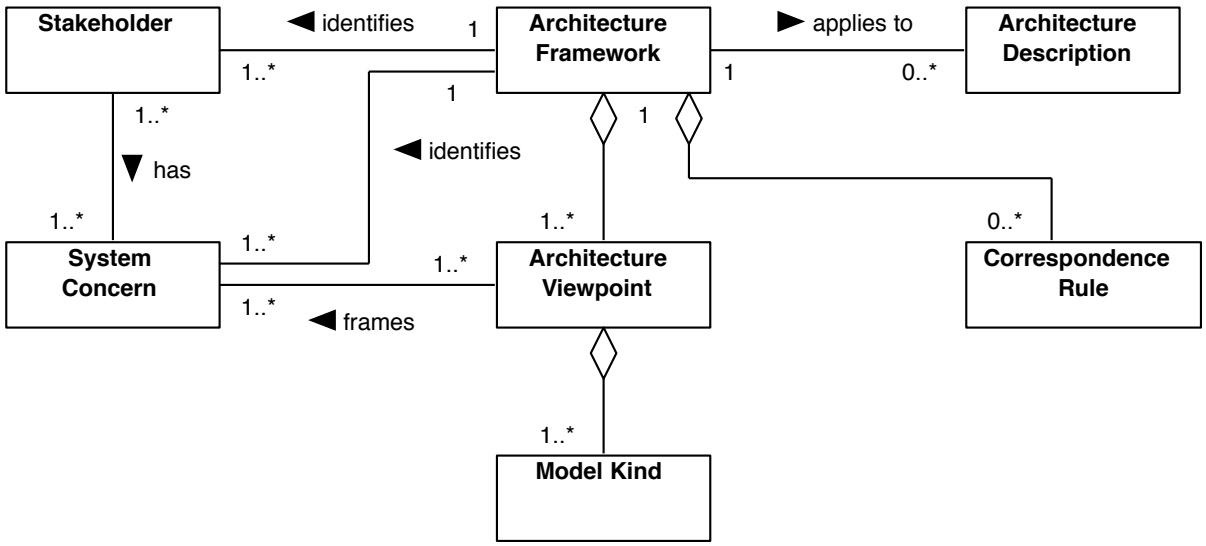


Figure 6.1: *Architecture framework (reproduced from ISO/IEC/IEEE 42010 (ISO/IEC/IEEE 2011))*

An *architecture framework* is a set of practices for architecture description used within a domain or community of stakeholders (ISO/IEC/IEEE 2011). A framework typically consists of a set of viewpoints for addressing recurring or typical concerns within that community. Figure 6.1 shows the metamodel for architecture frameworks from ISO/IEC/IEEE 42010 (ISO/IEC/IEEE 2011).

In this chapter, we present a documentation framework for architecture decisions which uses the conventions of ISO/IEC/IEEE 42010. It comprises all elements defined in Figure 6.1. The four viewpoints of the framework were successively developed to frame the concerns described in the previous section. Each of the viewpoints is dedicated to concerns that are not, or not sufficiently framed by the previously created viewpoint. Starting from the decision detail viewpoint, which mainly addresses concerns related to the rationale behind decisions (C3-C6); we defined the decision relationship viewpoint, which focusses on concerns pertaining to relationships between decisions (C8-C15). The decision stakeholder involvement viewpoint allows to explicate the relationships between stakeholders and decisions (C16-C19). Finally, the decision chronology viewpoint was developed to satisfy the remaining temporal concerns in decisions (C20-C22). Apart from the key concerns mentioned here, each viewpoint addresses additional concerns that will be described in the following subsections.

In the remainder of this section, we outline the four viewpoints and show example views. A thorough definition of the framework can be found in Appendix B.3.

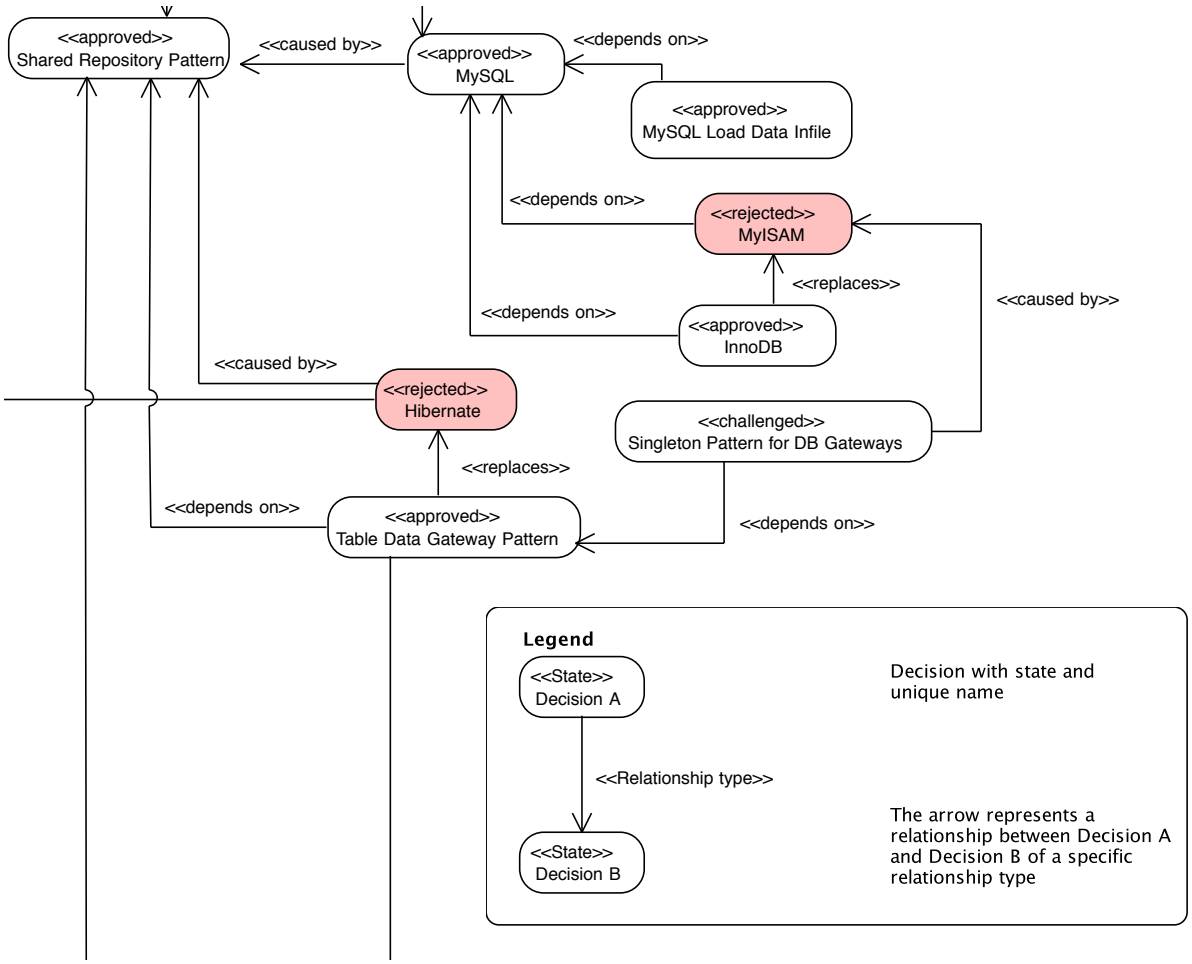


Figure 6.2: *Detail of a relationship view*

Table 6.3: *Typical stakeholders and concerns for the decision relationship viewpoint*

Stakeholders	Concerns
Architects	C1,C2, C8, C9, C10, C11, C12, C13, C14, C15, C22, C23
Reviewers	C1, C2, C9, C10, C11, C12
Domain experts	C23

6.3.1 Decision relationship viewpoint

The decision relationship viewpoint makes relationships between architecture decisions explicit. It shows architecture decisions, their relationships to other decisions, and their current states. It has no temporal component, i.e., it shows a snapshot of the system in a particular moment in time. Typical stakeholders for this viewpoint are architects, reviewers and domain experts. Table 6.3 shows the concerns framed by the viewpoint, as related to the mentioned stakeholders. They center around impact, dependency and relationship analysis. Additionally, relationship views are well-suited for getting an overview of all decisions made. Please refer to Table 6.1 for the descriptions of the

concerns.

Figure 6.2 shows a detail from a relationship view that was created in a preliminary study, conducted to test the decision viewpoints. The preliminary studies are further described in Section 6.4.

6.3.2 Decision stakeholder involvement viewpoint

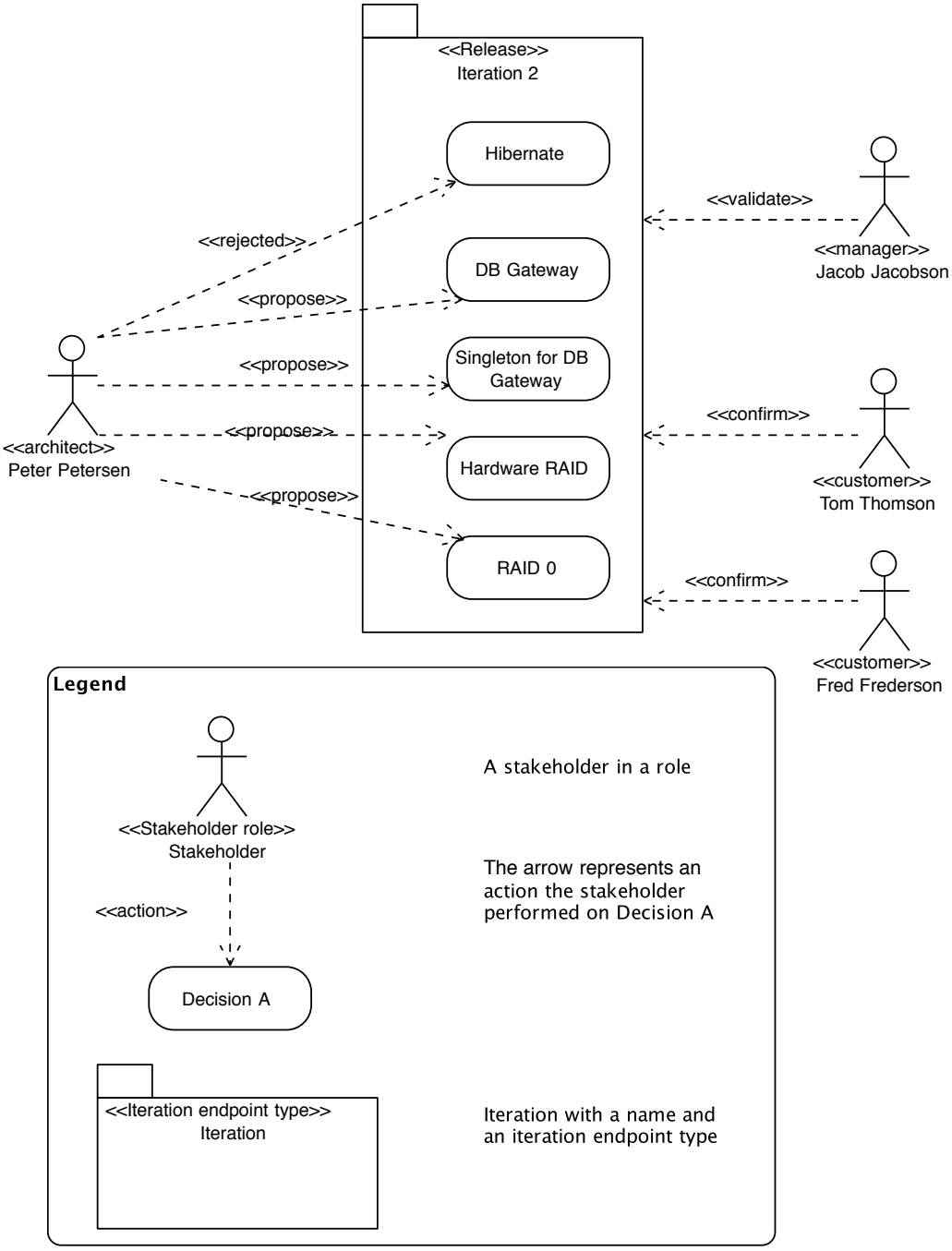


Figure 6.3: *Example stakeholder involvement view*

Table 6.4: *Typical stakeholders of the decision stakeholder involvement viewpoint*

Stakeholders	Concerns
Reviewers	C16, C17, C18
Architects	C1, C16, C17, C18, C19
Managers	C18,C19

The decision stakeholder involvement viewpoint shows the responsibilities of relevant stakeholders in the decision-making process. Views resulting from this viewpoint have no temporal component. They show decisions, actions and stakeholders involved in the decision-making process, within one specific architecture iteration. This information is important with regard to personalization of architectural knowledge, i.e., documenting not the knowledge per se, but "who knows what". For many reasons, in some projects, it is not feasible to fully document the rationale behind all architecture decisions. Other knowledge remains tacit; it is not documented at all. In these situations, the rationale remains in the heads of the people who were involved in the decision making process. Stakeholder involvement views make these involvements explicit. Furthermore, the viewpoint allows to analyze the impact of personnel on the success or failure of a project. If, for instance, a large number of decisions made by one specific architect were rejected, then this could be an indicator for a problem. As a side effect, explicitly documenting responsibilities creates accountability, in that people assume responsibility for the decisions they are involved in. On the other hand, this might cause architects to neglect the usage of stakeholder views, because they fear accountability.

Typical stakeholders for this viewpoint are reviewers, architects and managers. Table 6.4 shows the concerns framed by this viewpoint, related to the respective stakeholders. They center around stakeholder involvement in decisions. Please refer to Table 6.1 for the descriptions of the concerns.

Figure 6.3 shows a detail from a stakeholder involvement view that was created in a preliminary study conducted to test the decision viewpoints. The names of the involved stakeholders were changed for privacy reasons. The preliminary studies are further described in Section 6.4.

6.3.3 Decision chronology viewpoint

The decision chronology viewpoint shows the evolution of architecture decisions in chronological order. Besides decisions, it shows architecture iterations and its endpoints, which can be further specified by a type and a date. The chronology viewpoint is the only proposed decision viewpoint that has a temporal component. Typical stakeholders for this viewpoint are reviewers, architects and new project members, who need to comprehend the architecting process during system evolution. Table 6.5 shows the concerns framed by this viewpoint related to the respective stakeholders. A chronology

Table 6.5: *Typical stakeholders of the decision chronology viewpoint*

Stakeholders	Concerns
Reviewers	C1, C2, C20, C21
Architects	C1,C2, C20, C21,C22
New project members	C20

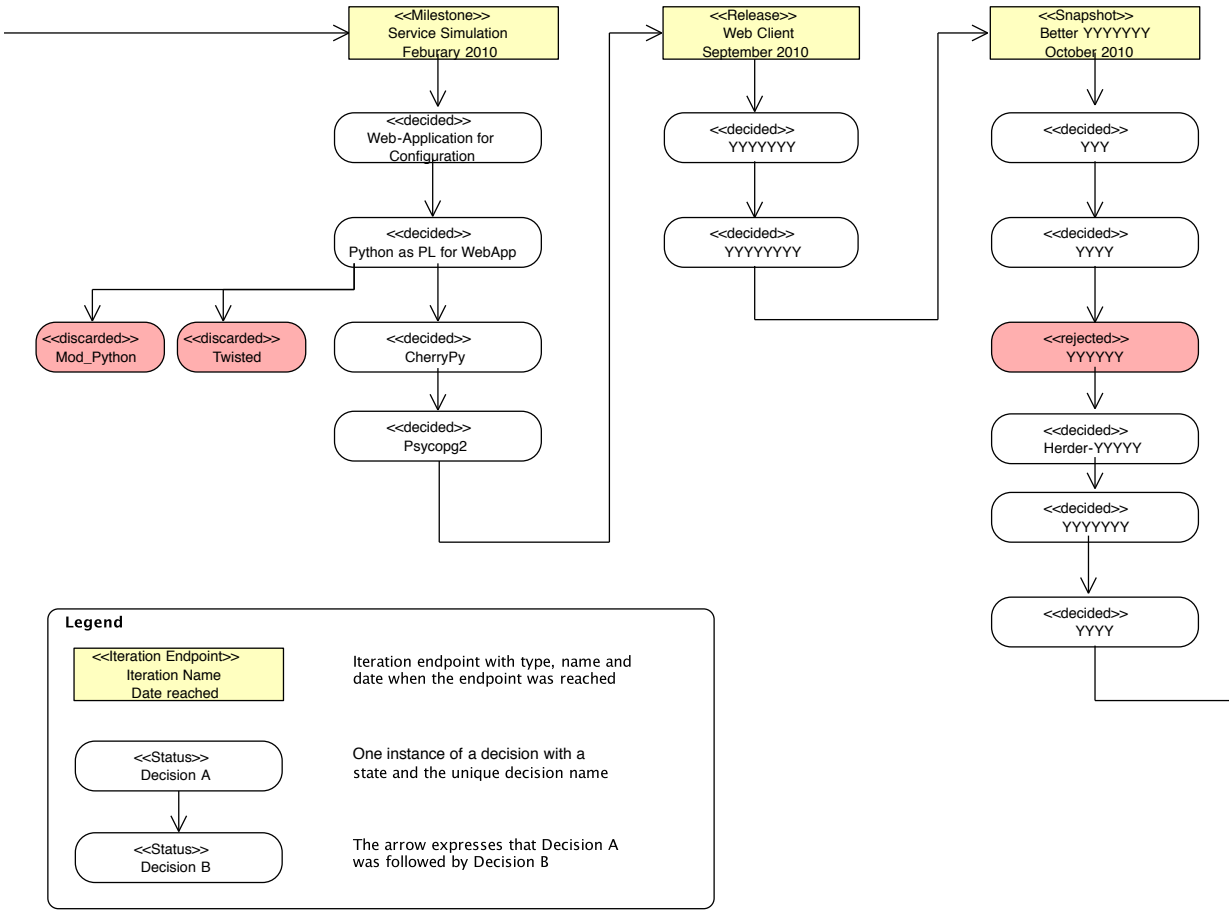


Figure 6.4: *Partially censored detail from a chronology view created during the case study*

view shows all versions of every architecture decision of a system. A version of an architecture decision is defined as a decision with a state. For instance, a decision that was *tentative*, then became *decided* and finally *approved* is represented with three instances in one chronology view.

Figure 6.4 shows a detail from a chronology view created in the case study, which is presented in Section 6.4.

Table 6.6: *Typical stakeholders of the decision detail viewpoint*

Stakeholders	Concerns
Reviewers	C1, C2, C3, C4, C5, C6, C11,C18
Architects	C1, C2, C3, C4, C5, C6, C11,C18
Customers	C3, C6
Managers	C18
New project members	C3
Requirements Engineers	C6

6.3.4 Decision detail viewpoint

Although textual decision descriptions have disadvantages, as mentioned in the introduction, they are certainly useful for grasping large parts of the rationale behind decisions. We propose to complement the previously described viewpoints, which only include partial information about the decisions, with a decision detail viewpoint. Each viewpoint frames specific stakeholder concerns, omitting information that is irrelevant for the respective stakeholders of those specific concerns. While the other viewpoints provide an overview over the decisions made and focus on the relationships between decisions, the decision detail viewpoint gives detailed information about single decisions.

Currently, there is no commonly accepted template in the literature to describe architecture decisions, although many proposals exist. Shahin et al. analyzed nine architecture decision models with respect to similar description elements (Shahin et al. 2009). They distinguish between major (e.g., decision outcome, related requirements, design options, arguments) and minor description elements (e.g., issue, decision group, state, related decisions, related artifacts, consequences and stakeholders). We decided to create our own template that contains all major elements plus selected minor elements that turned out to be useful in three pilot studies, we conducted to test the decision viewpoints. The resulting set of description elements is:

- **Name**: A short name of the decision that serves as a key in the other views
- **Current State**: The current state of the decision. Please refer to Figure B.6 for a list of all possible decision states.
- **Decision Groups:** A decision can be associated to one or more groups, which share specific characteristics. Decisions could for instance be grouped by subsystem, architecture team who made the decision, or quality attribute requirements. The concept of a decision group is equal to the group concept in Tyree and Akerman's decision template (Tyree and Akerman 2005), and the decision categories in Kruchten's ontology (Kruchten 2004a).

Name	**MySQL Master Slave**
Current Version	3 (MS2 <<Release>>)
Current State	Approved
Decision Group	None
Problem/Issue	The physical storage on the database server uses a RAID 0 configuration. If one of the discs in the array fails, a complete loss of data would be the effect. This violates the reliability requirements.
Decision	Provide a second hardware node that runs MySQL in slave configuration. The primary database server is configured as master.
Alternatives	Periodically backup the whole database as complete image of the server
Arguments	With a master slave configuration, all changes made to the master are automatically synchronized with the slave server. If the master fails or needs to be maintained, the slave can be reconfigured to act as a master within 30 seconds. A backup would only capture snapshots and recovery would take much longer.
Related decisions	• This <<caused by>> RAID 0 • This <<caused by>> MySQL DBMS
Related requirements	NFR1, NFR2, NFR3, NFR4
History	**Stakeholder** / **Action** / **Status** / **Iteration** F. Fredson <<Architect>> / <<Propose>> / <<Tentative>> / MS1 E. Ericson <<Architect>> / <<Validate>> / <<Decided>> / MS1 T. Thompson <<Reviewer>> / <<Confirm>> / <<Approved>> / MS2

Figure 6.5: *Example detail model of an architecture decision*

- **Problem/Issue**: The circumstances, under which the architect felt the need to make a decision among one or more alternatives. In other words: the issue addressed by the decision.

- **Decision**: The outcome of the decision. In other templates, this element is called *solution*.

- **Alternatives**: The alternative solutions considered when making the decision.

- **Related decisions:** All decisions that have a relationship to the decision. The available relationship types are defined in B.3.2.

- **Related requirements**: The decision detail viewpoint is currently the only viewpoint that allows to trace architecturally significant requirements and architecture

decisions. An additional viewpoint that specializes on traceability between requirements and decisions will be discussed in Section 6.6.

- **History**: The history of the described decision. The history contains all state changes, i.e., when the decision was proposed, decided, approved and so on.

Typical stakeholders for this viewpoint are reviewers, architects, customers, managers, new project members and requirements engineers. Table 6.6 shows the concerns framed by this viewpoint in relation to the respective stakeholders.

Writing elaborate decision descriptions is a resource-intensive task. However, the flexibility of this viewpoint allows companies to document just as much as needed for their individual purposes. Some organizations might even decide to skip this viewpoint completely and only use some of the other proposed viewpoints to document key aspects of their decisions. Others might decide to use a subset of the proposed elements of our template. The project team in our case study felt that there is no need to document every decision in the same level of detail. They described the major and most important decisions in detail, while putting less effort in describing minor decisions.

6.4 A case study

To validate the usage of the presented architecture decision viewpoints in a real software project, we conducted a single case, embedded case study (Gray 2009). In a *single case* design, only a single case is observed; *embedded* refers to the fact that multiple units of analysis are observed in the case.

This case study is a project executed at the Institute for Internet-Security (IFIS). The IFIS is a German organization in the Internet and network-security domain. We examined the decision viewpoints, as applied to a software project called "Sandnet" (Rossow et al. 2011). Sandnet is a system that executes malware like viruses, worms and bots in a controlled environment to analyze their network behavior.

6.4.1 Study goal, research questions and variables

The goal of the case study is to explore whether the architecture decision viewpoints effectively support software architecture activities. To derive concrete research questions, we explained the decision viewpoints to the architects of the Sandnet project and let them decide for which architecture activities they could be used in their project. They identified the following activities: general architecture decision documentation, communication between stakeholders, technical architecture reviews, and reusing architecture decisions in other projects.

Then the architects of Sandnet expressed their concerns in decision documentation with respect to these activities. These concerns were mapped to the list of concerns shown in Table 6.1 and supplemented by concerns that the authors found important.

Table 6.7: *Architecture activities, related concerns and viewpoints*

Architecture activity	Concerns	Viewpoints
A1 - AD documentation	All	All
A2 - Stakeholder communication	C1, C2, C3, C4, C5, C6, C8, C9, C10, C11, C12, C17, C19, C20, C21	All
A3 - Technical architecture reviews	C1, C2, C3, C5, C6, C7, C8, C9, C10, C11, C12, C16, C17, C18, C20, C21	All (C7 not covered)
A4 - Reusing ADs	C3, C5, C8, C17, C20, C23	Relationship viewpoint, chronology viewpoint, details viewpoint

Finally, the authors assigned all viewpoints to the activities that were designed to frame at least one of the concerns mentioned by the architects. Table 6.7 shows the results.

Based on the information shown in Table 6.7, the architects decided to create views according to the decision relationship viewpoint, the decision chronology viewpoint, and the decision detail viewpoint. They did not see additional benefit in documenting a stakeholder involvement view. The project was small enough to remember the involvement of all stakeholders. This is in-line with ISO/IEC/IEEE 42010, which propagates the choice of viewpoints to architects using the standard, according to the prioritization of the concerns. The concerns C16, C17, and C19 are not covered in the case study, because they are exclusively satisfied by the stakeholder involvement viewpoint. Concern C7 (*What decisions have conflicting impacts on concern C?*) is currently not covered by any of the viewpoints in the framework. We reflect on this issue in Section 6.6.

Next, we formulated concrete research questions matching the architecture activities selected by the architects and the related viewpoints.

The four research questions are summarized in Table 6.8 and discussed in the remainder of this section.

RQ1 - What is the effort of documenting architecture decisions using architecture decision viewpoints?

Research question one (RQ1) is about the effort that architects have to make, in order to document architecture decisions using decision viewpoints. This question is more specific than the question "Do decision views effectively support stakeholders to document architecture decisions?", as could be derived from the main research goal applied to activity A1. We made RQ1 more specific, because the effort is essential to judge the effectiveness of the decision viewpoint approach, and it can be explicitly measured. RQ1 can be refined with respect to the different viewpoints under study, i.e.,

Table 6.8: *Architecture activities and related viewpoints*

Code	Research Question	Viewpoints
RQ1	What is the effort of documenting architecture decisions using architecture decision viewpoints?	Relationship VP, Chronology VP, Details VP
RQ2	Do decision views effectively support stakeholders to understand the architecture?	Relationship VP, Chronology VP, Details VP
RQ3	Do decision views effectively support architecture reviews?	Relationship VP, Chronology VP, Details VP
RQ4	Do decision views support architects to distill reusable decision subgraphs?	Relationship VP, Chronology VP, Details VP

- What is the effort of creating a decision relationship view that conforms to the decision relationship viewpoint?
- What is the effort of creating a decision chronology view that conforms to the decision chronology viewpoint?
- What is the effort of creating a decision detail view that conforms to the decision detail viewpoint?

One dependent variable is defined for RQ1: the effort of creating each of the views is measured in person-hours, a common unit to express effort in software development projects, here defined as the number of hours spent by one person. Table 6.9 summarizes the variables. The statistical scale used to measure values for the variable is a ratio scale, which means that the possible values for the variable are ordered, have a zero point, and have equal intervals (Wohlin et al. 2012). The scale type is needed to determine which statistical calculations apply for the variable. The range shows which values can actually be assigned to the variable. In this case, the time is measured in whole hours; thus, the variable can take positive natural numbers of hours including zero hours.

Table 6.9: *Dependent variables of RQ1*

Description	Scale Type	Unit	Range
Time spent to create view	Ratio	Person-hours	Positive natural numbers including zero

Table 6.10 shows potential variables that might have an influence on the effort needed to document the views. These independent variables relate to characteristics

of the architects who used the viewpoints to create views, and to the characteristics of the software project that was documented. It is in the nature of case studies that these variables cannot be controlled (Gray 2009), thus we describe them thoroughly so readers can use them to judge the external validity of the results. This and other potential threats to validity are discussed in Section 6.4.4.

Table 6.10: *Independent variables of RQ1*

Description	Scale Type	Unit	Range
Time the architects have worked in the IT industry	Ratio	Years	Positive natural numbers including zero
Time the architects have worked as software designers/architects	Ratio	Years	Positive natural numbers including zero
Number of architects who created the views	Ratio	Persons	Positive natural numbers including zero
Duration of the documented project	Ratio	Months	Positive natural numbers including zero
Project size	Ratio	Person-Months	Positive natural numbers including zero
Number of made decisions	Ratio	Decisions	Positive natural numbers including zero
Average number of words used to document one decision in the detail view	Ratio	Words	Positive natural numbers including zero

RQ2 - Do decision views effectively support stakeholders to understand the architecture?

In research question two (RQ2), we investigate, if decision views, corresponding to the decision viewpoints, support stakeholders to understand the architecture. RQ2 is decomposed with respect to the different viewpoints under study:

- Do decision relationship views effectively support stakeholders to understand the architecture?
- Do decision chronology views effectively support stakeholders to understand the architecture?
- Do decision detail views effectively support stakeholders to understand the architecture?

The level of understanding of an architecture that stakeholders gain after studying the decision views is hard to measure and especially hard to quantify. As dependent variable, we estimate the level of understanding by qualitatively analyzing questions asked and comments expressed by stakeholders to the architects after having studied the decision views (see Table 6.11).

Table 6.11: *Dependent variables RQ2*

Description	Scale Type	Unit	Range
Level of architecture understanding by the stakeholders	n.a.	Open	Open

Table 6.12 shows independent variables that could influence the dependent variables. They relate to characteristics of the stakeholders who studied the views and the characteristics of the software project that was documented.

RQ3 - Do decision views effectively support architecture reviews?

Research question three (RQ3) aims at finding out if views, corresponding to the decision viewpoints, support activities performed during architecture reviews. According to (IEEE 2008), a review is an evaluation of a software product by a team of qualified personnel. Accordingly, an architecture review is an evaluation of the software architecture by stakeholders who are either domain experts or architecture experts. In the case study, we had the opportunity to observe the usage of decision views in an architecture review. Details on the architecture review are given in Section 6.4.3.

Unfortunately, the IFIS organization had not been following an established or systematic review approach before the case study. The architects of the Sandnet project previously performed reviews in an ad-hoc manner, without involving other stakeholders and without systematically documenting review outcomes. Thus, the effect of using decision views in the review cannot be compared to the previous practice.

RQ3 is decomposed with respect to the different viewpoints under study:

- Do decision relationship views effectively support architecture reviews?
- Do decision chronology views effectively support architecture reviews?
- Do decision detail views effectively support architecture reviews?

Two dependent variables are defined for RQ3. The level of support for the review activities is estimated by qualitatively analyzing the transcript from a focus group, conducted with the participants of an architecture review performed as part of the case study. We also measure the number of risks that came up during the review (see Table 6.13). An elaboration of the focus group can be found in Section 6.4.2.

Table 6.12: *Independent variables RQ2*

Description	Scale Type	Unit	Range
Time the stakeholder has worked in the IT industry	Ratio	Years	Positive natural numbers including zero.
Time the stakeholder has worked as software designer/architect	Ratio	Years	Positive natural numbers including zero.
Time the stakeholder has worked in the network security domain	Ratio	Years	Positive natural numbers including zero.
How often has the stakeholder been involved in the analysis of architecture decisions	Ordinal	n.a.	Five point Likert-scale. One for very frequently, Five for very rarely.
Time the stakeholders took to study the views	Ratio	Minutes	Positive real numbers including zero.
Duration of the documented project	Ratio	Months	Positive natural numbers including zero.
Number of made decisions	Ratio	Decisions	Positive natural numbers including zero.
Average number of words used to document one decision in the detail view	Ratio	Words	Positive real numbers including zero.
Time spent to study a view	Ratio	Person-hours	Positive natural numbers including zero.

Table 6.13: *Dependent variables RQ3*

Description	Scale Type	Unit	Range
Level of support for the review activities	n.a.	Open	Open
Number of risks uncovered during the review	Ratio	Risks	Positive natural numbers including zero.

Table 6.14 shows independent variables that could influence the suitability of decision views to support architecture reviews. They relate to characteristics of the people who took part in the review, the software project, and the review approach that is being followed.

Table 6.14: *Independent variables RQ3*

Description	Scale Type	Unit	Range
Time the reviewers have worked in the IT industry	Ratio	Years	Positive natural numbers including zero.
Time the reviewers have worked as software designer/architect	Ratio	Years	Positive natural numbers including zero.
Time the reviewers have worked in the network security domain	Ratio	Years	Positive natural numbers including zero.
How often have the reviewers been involved in the analysis of architecture decisions	Ordinal	n.a.	Five point Likert-scale. One for very frequently, Five for very rarely.
How often have the reviewers been involved in architecture reviews	Ordinal	n.a.	Five point Likert-scale. One for very frequently, Five for very rarely.
Activities performed in the architecture review	n.a.	Open	Open
Duration of the documented project	Ratio	Months	Positive natural numbers including zero.
Number of decisions documented	Ratio	Decisions	Positive natural numbers including zero.
Average number of words used to document one decision in the detail view	Ratio	Words	Positive real numbers including zero.

RQ4 - Do decision views support architects to distill reusable decision sub-graphs

The last research question (RQ4) is about identifying a set of decision sub-graphs or logically grouped architecture decisions that can be reused as a whole in other software projects. An example for such a decision sub-graph is the choice of a database management system (DMBS), the choice of a hardware platform for the DBMS, the choice of an operating system for the hardware platform, and a communication protocol for accessing the DBMS. The sub-graph contains all possible design options, the chronological order of the decisions and the rationale behind each of the decisions. RQ4 is decomposed with respect to the different viewpoints under study:

- Do decision relationship views support architects to distill reusable decision sub-graphs?

- Do decision chronology views support architects to distill reusable decision sub-graphs?
- Do decision detail views support architects to distill reusable decision sub-graphs?

To find out, if decision views support this process, we independently asked the architects and other technical stakeholders to identify concrete reusable decisions. Subsequently, we counted the decisions and evaluated the level of support provided by the views in order to identify them (see Table 6.15).

Table 6.15: *Dependent variables RQ4*

Description	Scale Type	Unit	Range
Number of identified reusable decisions	Ratio	Decisions	Positive natural numbers including zero.
Level of support for identifying reusable decision sub-graphs	n.a.	Open	Open

Table 6.16 shows independent variables that might have an influence on the dependent variables. They relate to characteristics of the software project, and the architects and technical stakeholders, who distilled the reusable decisions (referred to as subjects in Table 6.16).

Table 6.16: *Independent variables RQ4*

Description	Scale Type	Unit	Range
Time the subject has worked in the IT industry	Ratio	Years	Positive natural numbers including zero.
Time the subject has worked as software designer/architect	Ratio	Years	Positive natural numbers including zero.
Number of made decisions	Ratio	Decisions	Positive natural numbers including zero.

6.4.2 Study design and execution

The aim of case studies in general is the investigation of contemporary phenomena in their natural context (Robson 2011, Yin 2003). According to Robson's classification scheme for empirical research purposes (Robson 2011), our case study is exploratory

in nature. We aim at understanding how and which architecture activities can be supported by decision viewpoints. We chose the case study method because we aim at validating the proposed documentation framework in industrial practice, where the researchers' control on the observed events is typically low. This is in line with Gray, who suggests that case studies are appropriate where "how" and "why" questions about a set of events must be answered, over which the researcher has no control (Gray 2009). Additionally, the effect of using decision viewpoints in a project might be multi-faceted, which also makes the case study method more suitable than other empirical research methodologies which require control over independent variables (e.g., a controlled experiment) (Wohlin et al. 2003).

To elicit hidden variables and as a preparation for the study design, we conducted three small pilot studies with the decision viewpoints. In all three projects, our viewpoints were used to document the architecture decisions made. One of the authors was involved in all of the projects.

- Open Pattern Repository (OPR): A freely usable, open source online repository for patterns and technologies. All project artifacts including source code, architecture decisions and design documents can be found in the project's Google code repository (University of Groningen, Software Engineering and Architecture Group 2012b).

- Open Decision Repository (ODR): The ODR is an open source web documentation tool for architecture and design decisions. Like the OPR, all project artifacts can be found in the project's Google code repository (University of Groningen, Software Engineering and Architecture Group 2012a). We elaborate on the Open Decision Repository in Section 6.6.

- Measurement collector for network traffic analysis: This so-called "raw data transfer system" is part of an Internet early warning suite. The part of the system we looked at is responsible for collecting data measured by probes that are located in different autonomous systems that comprise the German connection to the Internet. Details about the vendor as well as the software itself cannot be provided, as the organization asked us to treat this data confidentially.

In the following subsections, the observed case and the used rchitecture dec are described.

Case description

The IFIS, in which the case study was conducted, currently has 49 employees working in nine main projects (as of March 2011). The project domains include cloud computing, botnet analysis, identity management and Internet early warning. Customers of the organization are, among others, large telecommunication providers and the German Federal Office for Information-Security.

In the project under study (Sandnet), malware collected on the Internet is executed in a prepared network for further analysis. The software provides a controlled execution environment for the extensive analysis and safe execution of malware samples. One of the major challenges in the project is to provide a realistic environment for malware execution on the one hand, while preventing the malware from doing harm in external networks. For instance, the Sandnet forwards denial-of-service or spam attacks to a dedicated honeypot server to protect the original destination of the attack, while analyzing the complete network traffic. The project started in September 2009 and is ongoing. In total, four developers are involved in the project; two of them are responsible for the software architecture. Important stakeholders of the system, apart from the developers and architects, are network administrators who operate the Sandnet in their networks and malware authors, who have a negative stake in the system, because they do not want their malware to be analyzed. The project team does not follow a pre-defined software development process. They work in small iterations of a few weeks and document the system in a company-wide wiki.

Data collection

Data gathered in case studies is mainly qualitative. Because qualitative data is typically less precise than quantitative data, it is important to use triangulation to increase the precision of the study (Runeson and Höst 2009). Triangulation provides a broader representation of the research object under study. We use two different types of triangulation: methodological and data source triangulation (Stake 1995). Methodological triangulation takes different types of data collection into consideration. In this case study, we used participant observation, focus group, interview and analysis of work artifacts (Lethbridge et al. 2005). Data source triangulation uses multiple data sources at potentially different occasions. During this case study, we collected data during multiple sessions with the architects of the project and other stakeholders. Table 6.17 shows an overview of the sessions, the data collection method used, the data sources, and related research questions.

In the following, the data collection methods used are described in more detail.

- **Participant observation**: Participant observation is a popular data collection method in case studies (Mack et al. 2005, Gray 2009, Yin 2003, Seaman 1999). It involves systematic viewing of actions performed by the observed subjects, recording, analysis and interpretation of the observed behavior. In the observation sessions listed in Table 6.17, one of the researchers joined the observed subjects during their work. The researcher had access to all documents, the subjects used during these sessions and he listened to entire communications in cases where multiple subjects collaborated. During these sessions, the researcher took notes about working topics, time spent, communication issues and other observations that could have a relation to the research questions and their corresponding variables. The session notes and copies of the project artifacts used by the observed subjects were stored in a case study database as proposed by Gray (Gray 2009).

Table 6.17: *Data collection methods*

Data Collection Method	Sessions	Data sources	RQs
Partic. observation	Initial architecture decision elicitation	Architect1, Architect2	RQ1
	Initial creation relationship view	Architect1	RQ1
	Refinement relationship view	Architect2	RQ1
	Discussion about relationship view	Architect1, Architect2	RQ1
	Initial creation of decision detail view, refinement relationship view	Architect1	RQ1
	Discussion of detail view, refinement relationship view, initial creation chronology view	Architect1, Architect2	RQ1
	Planning of architecture review based on decision views	Architect1, Architect2	RQ3
	Revision decision views, revision of architecture review planning	Architect1	RQ1, RQ3
	Architecture review	Architect1, Architect2, three domain experts, two architecture experts	RQ2, RQ3
Focus group	Conducted immediately after the architecture review	Architect1, Architect2, three domain experts, two architecture experts	RQ2, RQ3, RQ4
Interview	Conducted at the end of the case study	Architect1	RQ1-RQ4
	Conducted at the end of the case study	Architect2	RQ1-RQ4
Analysis of work artifacts	n.a.	Existing architecture documentation, produced decision views, outcome of the architecture review	RQ1, RQ2, RQ4

- **Focus Group**: Focus group data collection is a well-documented technique that assembles small groups of peers to discuss particular topics. Discussion in focus groups is largely open, but it is directed by a moderator allowing soft, or qualitative issues to be explored. Kontio et al. mention additional advantages of focus groups in comparison to other qualitative research methods (Kontio et al. 2008). They observed that the interactive nature of the group discussions with people from different backgrounds encourages participants to react to the comments made by other participants, thus reflecting and building on each other's experiences. It also helps to validate comments and positions, as some points made by participants may result in other participants confirming similar, almost similar or opposite points. These insights would be invisible in personal interviews. The researchers have experience in conducting focus groups from previous studies; therefore a pilot session was not performed. The guidelines presented in (Mack et al. 2005) were used to prepare and conduct the focus group. In particular, a question guide was created in advance and internally reviewed by the authors. The questions in a question guide are not asked directly, but serve as orientation for the moderator of a focus group. Focus groups are most productive, if the participants are encouraged to have an open discussion, while the moderator tries to lead the discussion in a way that all important questions are answered. The used question guide can be found in Appendix B.5.

  The focus group was conducted by involving all people who took part in a technical architecture review; one of the authors was allowed to join that review. The complete session was audio-recorded with the participants' consent and afterwards transcribed. Additionally, the researcher took notes about observations that could not be captured on tape, e.g., collective nodding of participants. The audio recording, notes and the transcript were stored in the case study database.

- **Interview**: Interviews allow researchers to gain in-depth knowledge about the interview topics. Like focus groups, they enable researchers to ask interviewees for clarification to solve potential misunderstandings (Lethbridge et al. 2005). Interviews are an appropriate means to collect opinions and impressions about the object under study (Seaman 1999). The two interviews with the architects lasted between 50 and 60 minutes each. They were conducted as videoconferences, because the architects were not on-site in their organization at that time. Both interviews were digitally recorded with the participants' consent and later transcribed. The original audio files and the transcripts were stored in the case study database.

- **Analysis of work artifacts**: This data collection method is used to uncover information about how the architects applied and used decision views by looking at their output. In this particular case, we looked at architecture documentation in wikis, the decision views created by the architects, the architecture review planning document and the architecture review report to gather data relevant for our research questions. All collected documents were stored in the case study

database. The authors of the documents were contacted to clarify questions and issues related to the documents.

6.4.3 Analysis

We use descriptive statistics and qualitative analysis to examine the data gathered during the case study. This section is sub-divided according to the research questions.

Analysis RQ1 - What is the effort of documenting architecture decisions using architecture decision viewpoints?

As described in the study design, the effort of documenting architecture decisions is influenced by some independent variables. Table 6.18 shows descriptive statistics for the variables defined in Table 6.10.

Table 6.18: *Independent variables RQ1*

Variable	Values
Time the architects have worked in the IT Industry	Architect1: 6 years, Architect2: 6 years
Time the architects have worked as software designers/architects	Architect1: 4 years, Architect2: 5 years
Number of architects who created the views	2
Duration of the documented project	13 months
Project size	21 person-months
Number of decisions documented	56
Average number of words used to document one decision in the detail view	61

The duration of the documented project is the total time in calendar months spent on the project, whereas the project size is the time spent in person-months. The variable *number of decisions documented* indicates the number of decisions that have been documented using the decision view approach. To the best of our knowledge, the architects documented every design decision they found architecturally significant in the project. The "architectural significance" of decisions, however, is not essential for the decision documentation approach presented here. Many definitions for architecture decisions exist; e.g., architecture decisions are those decisions that have an impact on the system's quality attributes; others emphasize on the external visibility of those decisions.

We keep it simple by assuming that a decision is architectural in nature, if the architect believes it is architectural and should be documented.

For the calculation of the average number of words used to document a decision, the words used in every decision detail model were counted; thus no other views were taken into account for this variable.

The effort, according to the dependent variable, is calculated in terms of time spent to create the different views. Table 6.19 shows the effort in person-hours spent to create views for the corresponding viewpoints. Decision elicitation is shown as an additional category. This is necessary, because architecture decisions have not been documented in the Sandnet project prior to the case study; thus, all architecture decisions had to be elicited from existing project documentation and brainstorming sessions by the two architects.

Table 6.19: *RQ1 - Effort for creating views*

Decision Elicitation	Relationship view	Detail view	Chronology View
11 person-hours	4 person-hours	7 person-hours	2 person-hours

The total effort for creating the three views according to our viewpoint definition was 24 person-hours, which equals 3 person-days and 0.14 person-months. The effort has to be analyzed in the context of the project size. In this case, the project size was 21 person-months, which equals 3696 person-hours (21 months * 22 days * 8 hours). Table 6.20 shows the effort for creating each view as percentage of the total development effort. In this calculation, the effort for decision elicitation was portioned equally to the three views, as it cannot be assigned to a single view.

Table 6.20: *Percentage effort for creating views*

View	Percentage of total development effort
Relationship View	0.21%
Chronology view	0.15%
Detail View	0.29%

In our experience, the order of view creation followed in the case study works well under the assumption that the relationship view is good for an initial documentation of decisions. It is a lightweight view in the sense that decisions can easily be added or removed and related to other decisions. In contrast to the decision detail view, in which decisions have to be described in textual form, revising decisions and relationships in the relationship view is just a matter of drawing ellipses and connectors. This allows for subsequent refinements in quick iterations. Once the main decisions and decision relationships are documented, effort can be invested to capture the decisions in the detail

view. Finally, the chronology view adds information about the evolution of the decisions. Remembering the correct chronological sequence of decisions requires a lot of reflection by the architects, if created after the fact. Therefore, in "green field" projects, it can make more sense to create the chronology view right away and revise it iteratively during the project.

The subjects in the case study had to overcome a learning curve, i.e., they learned how to use the viewpoints in order to design the views. We assume that the same subjects would be able to document decision views more quickly in future projects. However, this assumption is subject to further empirical evaluation.

The total effort for creating the three views was approximately 0.65% of the total development effort in person-time. In addition to the quantitative analysis of the effort needed to create decision views, we asked the architects about their subjective estimation of the effort. Moreover, we wanted to know which views they would create if they were very limited in time.

The transcripts of the interviews with the architects were analyzed using the *constant comparative method* (Glaser and Strauss 1967), a well-established theory generation method in qualitative analysis (Seaman 1999). In detail, the following procedure was followed. The original comments in the interviews were given mainly in colloquial speech and many of them can only be interpreted in the context of the whole interview. We browsed the transcripts of the interviews and searched for passages of text related to the research question. The respective passages were labelled and later grouped into patterns expressing their content as more formal, context-free statements. This procedure was used for the qualitative analysis of the interviews and the focus group conducted after the review. An example of the analysis process is given in Appendix B.4.

Finally, the original transcript, the extracted comments, and the derived statements were given to two of the participants of the focus group for validation. They concordantly acknowledged that the information in the derived statements is similar to the information in the original comments.

The following statements were derived from text passages in the interviews with the architects, labelled with RQ1:

- How reasonable the effort for creating decision views is, depends on the number of team members and the duration of the project.
- The decision detail view is the most important view for the original architects.
- The decisions in the detail view can be documented with just a few attributes (rather than all of them).
- The decision detail view is not helpful in isolation. It should be complemented at least with a relationship view in order to create an overview of decisions for the other stakeholders who were not directly involved in the decision making process.
- When there is no time to document all views, the architects would skip the chronology view and only create a detail view and a relationship view.
- Important decisions should be documented in more detail than less important decisions.

- The effort for creating decision views is definitely reasonable from the point of view of the project sponsor (the organization funding the project).
- The effort for creating decision views is marginal compared to the whole development effort of the project.

Analysis RQ2 - Do decision views effectively support stakeholders to understand the architecture?

As with RQ1, we use descriptive statistics to describe the independent variables. Table 6.21 shows descriptive statistics for the variables defined in Table 6.12.

Table 6.21: *Independent variables RQ2*

Variable	Statistics
Time the stakeholders have worked in the IT Industry	Stakeholder1: 12 years Stakeholder2: 5 years Stakeholder3: 15 years Stakeholder4: 2 years
Time the stakeholders have worked as software designers/architects	Stakeholder1: 12 years Stakeholder2: 5 years Stakeholder3: 0 years Stakeholder4: 2 years
Time the stakeholder has worked in the network security domain	Stakeholder1: 6 years Stakeholder2: 0 years Stakeholder3: 14 years Stakeholder4: 2 years
How often have the stakeholders been involved in the analysis of architecture decisions	Stakeholder1: Rarely Stakeholder2: Frequently Stakeholder3: Very rarely Stakeholder4: Rarely
Time the stakeholders took to study the views	Stakeholder1: 70 minutes Stakeholder2: 90 minutes Stakeholder3: 35 minutes Stakeholder4: 60 minutes
Duration of the documented project	13 months
Number of documented decisions	56
Average number of words used to document one decision in the detail view	61

The support for architecture understanding provided by the views is estimated in terms of the level of architecture understanding gained by the stakeholders after studying the views. The analysis of this dependent variable is done qualitatively. During the

case study, a technical architecture review was conducted with the architects, a few experts in the network security domain, and experts on software architecture. All people, except for the architects, were not familiar with the Sandnet system at all. Two days before the review, all participants were asked to analyze the system by studying the decision views we provided to them. Right after the review, we interviewed all participants in a focus group, where we also asked questions about the suitability of the views for understanding the system. The transcript of the audio-recorded focus group is used as a basis for the analysis. We used the constant comparative method, as described in Section 6.4.3. The following lists show the resulting participants' statements for the different viewpoints under study:

Relationship viewpoint:

- Relationship views clearly illustrate relationships between decisions.
- Relationship views support impact analysis.
- Relationship views illustrate decision relationships better than decision detail views.
- Relationship views illustrate dependencies between decisions.
- Relationships views do not contain enough information about a single decision.
- A combination of the decision detail view and the other decision views is necessary.
- The relationship view is good to get an overview of decisions made.
- The relationship view helps to start understanding a system, much better than the decision detail view.

Chronology viewpoint:

- The chronology view helps to understand the evolution of the system.
- Chronology views provide insights into the reasoning process of the architects.
- Chronology views show which solutions were considered, rejected and chosen.
- The chronology view is well suited to understand the decision making process in complex software systems.
- Chronology views are good to analyze change of the architecture over time.

Detail Viewpoint:

- Decision Detail views are important to analyze the reasoning and the details of every decision.
- A combination of the decision detail view and the other decision views is necessary.
- Detail views are hard to handle in isolation, because they produce large amounts of text on too many pages.

All Viewpoints:

- Decision views help to recap the decision making process for a single decision, including considered alternatives.

- Decision views are helpful to communicate architecture decisions to project teams taking over the system.
- Decision views prevent new project teams or team members from making fatal decisions.
- Decision views help people to understand which decisions were made for which reasons and which decisions were explicitly not made for specific reasons.
- Decision views are very helpful.
- Decision views are a good means to transfer architecture knowledge.
- The architects were amazed how much the stakeholders knew about the system after having studied the views.
- Software engineers should be obliged to create decision views as a complement to the other architecture documentation.
- Decision views capture architectural knowledge that cannot be recovered from traditional views; especially discarded and rejected decisions and the reasoning behind decisions.

Analysis RQ3 - Do decision views effectively support technical architecture reviews?

Table 6.22 contains descriptive statistics for the independent variables described in the study design.

In addition to the variables described in Table 6.22, the activities performed in the architecture review have an effect as independent variables. In the case study, a technical architecture review was performed based on a custom architecture review method, which will be explained in the following. The review was performed in two phases. In phase one, the reviewers received a review-planning document containing the schedule of the review, a description of the Sandnet project, the main stakeholders, and architecturally relevant requirements, as well as a network topology view and all documented decision views (i.e., a relationship view, a chronology view and a decision detail view). Additionally, they received a description of five technical scenarios. The scenarios had been documented in the company wiki by the architects as possible future changes or enhancements, prior to the case study. An example of a technical scenario, as described in the review-planning document is:"Currently, malware samples are executed by a sandpuppet for exactly one hour. How is the architecture of the system affected, if malware samples are executed for a complete day/week or even longer? Currently, there are 80 virtual machine slots for parallel execution of malware samples available". The reviewers were asked to study the views with respect to the scenarios in advance and write down all uncertainties and questions. Phase one was performed individually and off-site. Phase two was the actual review conducted on-site in the organization. Table 6.23 shows the schedule of the review and the activities performed.

During the review, the reviewers selected three scenarios out of the five available ones, based on their own judgement of the scenarios' importance. The detail view and the relationship view were used to identify and analyze decisions that have an effect on the respective scenario. The third selected scenario was skipped because of time

Table 6.22: *Independent variables RQ3*

Variable	Statistics
Time the reviewers have worked in the IT Industry	Stakeholder1: 12 years Stakeholder2: 5 years Stakeholder3: 15 years Stakeholder4: 2 years
Time the reviewers have worked as software designers/architects	N: 4 Stakeholder1: 12 years Stakeholder2: 5 years Stakeholder3: 0 years Stakeholder4: 2 years
Time the reviewers have worked in the network security domain	N: 4 Stakeholder1: 6 years Stakeholder2: 0 years Stakeholder3: 14 years Stakeholder4: 2 years
How often have the reviewers been involved in the analysis of architecture decisions	N: 4 Stakeholder1: Rarely Stakeholder2: Frequently Stakeholder3: Very rarely Stakeholder4: Rarely
How often have the reviewers been involved in architecture reviews	Stakeholder1: 15 times Stakeholder2: 5 times Stakeholder3: 1 time Stakeholder4: 3 times
Duration of the documented project	13 months
Number of decisions documented	56
Average number of words used to document one decision in the detail view	61

constraints. The participants planned an additional review session outside the time-period of the case study.

The support for technical architecture reviews provided by the decision views is estimated in terms of the number of risks that came up during the review and the level of support for the performed reviewing activities. To estimate the number of identified risks during the architecture review, we analyzed the risk evaluation forms filled in by the reviewers during the scenario-based reviews. Risk evaluation forms are part of the Software Risk Evaluation Method (SRE) (Williams et al. 1999) defined by the Software Engineering Institute, which was used by the architects to evaluate the risks uncovered during the review.

Table 6.23: *Review schedule*

Time	Activity
14:10	Start
14:20 - 14:30	Introduction of the Sandnet project by the architects
14:30 - 14:40	Introduction of the review process and goals by the review organizer
14:40 - 14:55	One of the architects explained the evolution of the system using the chronology view
14:55 - 15:05	Choice of three scenarios out of the five scenarios described in the review-planning document
15:05 - 15:50	Review scenario 1 using the decision detail view and the relationship view
15:50 - 16:30	Review scenario 2 using the decision detail view and the relationship view
16:30 - 17:00	Wrap-up session including discussion and documentation of all discovered issues

In total, the four reviewers recorded 27 distinct risks (reviewer 1: 3 risks, reviewer 2: 4 risks, reviewer 3: 12 risks, reviewer 4: 8 risks). Out of the 27 risks, the architects regarded five risks as high or medium severe. The other risks had either low severity, or the architects did not share the reviewer's opinion. The analysis of the support for reviewing activities was done based on an examination of the focus group transcripts, in which the support for architecture reviews was explicitly discussed. For the qualitative analysis, we followed the same procedure as described for RQ1. The following lists show the resulting statements assigned to the respective reviewing activities:

Activity: Architect explains evolution of the system using the chronology view

- The chronology view helps the architect to explain the evolution of the system.
- The chronology view helps to explain and remember the change of decisions over time.
- The chronology view helps reviewers to understand the evolution of the system.

Activity: Architect clarifies questions with respect to the system

- Decision views are well suited for explaining the architecture to stakeholders.
- Decision views can make sure that nothing is forgotten when explaining the architecture.
- Decision views make sure that the architecture can be communicated in a structured and understandable way.

Activity: Review scenarios

- The logical groups in the relationship view help to structure the decisions.
- The logical groups in the relationship view help to keep the overview over decisions.
- The relationship view supports the architect to perform impact analyses.
- The chronology view can be used to analyze changes between architecture iterations.
- The chronology view allows looking up changes between iterations quickly.
- Relationship views help reviewers to identify critical issues in the architecture.
- Relationship views visualize decision alternatives and allow reviewers to evaluate the final choice among the alternatives.
- Without decision views, the decisions and their relationships have to be elicited during the review.
- Decision views are a good basis for architecture reviews.
- Relationship views emphasize central decisions and decision alternatives.
- Relationship views allow identifying critical decisions quickly.
- Decision views capture architectural knowledge that cannot be recovered from traditional views. Especially discarded and rejected decisions and the reasoning behind decisions.
- The chronology view helps to understand changes between architectural milestones.
- Decision views allow reviewers to reassess if the architects have evaluated the right decision alternatives soundly.
- Decisions views can be used to assess if architectural problems were analyzed correctly.
- Decision views are helpful as a complement to traditional architecture documentation.

Analysis RQ4 - Do decision views support architects to distill reusable decision sub-graphs?

As for the fourth research question, we present descriptive statistics for the independent variables. Table 6.24 shows the results.

The support that decision views provide for distilling reusable decision sub-graphs is estimated in terms of the number of concrete reusable decision sub-graphs identified by the stakeholders and architects; and the level of support the decision views provide for identifying reusable decision sub-graphs. The latter is analyzed based on the transcripts created from the focus group after the review, in which we asked the participants to identify reusable decisions paths with the help of the documented views and the transcripts from the interviews with the architects.

The architects and stakeholders found the relationship view helpful to identify reusable decision sub-graphs. The relationship view documented for the Sandnet project contains six logical decision groups with 56 decisions in total. The stakeholders

Table 6.24: *Independent variables RQ4*

Variable	Statistics
Time the stakeholders have worked in the IT Industry	Stakeholder1: 12 years Stakeholder2: 5 years Stakeholder3: 15 years Stakeholder4: 2 years
Time the stakeholders have worked as software designers/architects	Stakeholder1: 12 years Stakeholder2: 5 years Stakeholder3: 0 years Stakeholder4: 2 years
Number of decisions documented	56

and architects were asked to identify concrete decision sub-graphs that could be reused in other software projects. From the six logical groups, they selected four groups that contain reusable decisions. Group one contains decisions about a web application used to access analysis data and configure sandpuppets; group two contains decisions related to the used database management system; the decisions in group three are about the control mechanism for the herders; and group four contains decisions related to the virtual machine technology and configuration used. The decisions in the remaining two logical groups were regarded as being too specific for being reusable. The analysis of the level of support was done qualitatively, similarly to the qualitative analysis described in RQ1, RQ2 and RQ3. The first set of statements was derived from comments made by the architects during the interviews, the second set of statements was derived from comments by the other stakeholders during the focus group after the review.

Architects:

- When identifying reusable decision sub-graphs, the dependencies upon other decisions can be evaluated using the relationship view.
- The relationship view provides good support for identifying reusable decision sub-graphs.
- Decisions that are strongly coupled to concrete requirements cannot be reused.

Stakeholders:

- Decision views are helpful to derive reusable decision sub-graphs.
- Logical decision groups in the relationship view are strong candidates for reusable paths.
- Logical decision groups in the relationship view are optimal decision combinations in the given context.
- The relationship view is well suited for extracting reusable architectural knowledge.

- Relationship views help to speed up the start of new projects.
- Relationship views provide guidance for architects with respect to dependencies between decisions and decision options.
- When reusing decisions from the relationship view, it is important to consider the context in which the decisions were made.
- Decision views help architects to recap the decisions made.
- For reusing decisions from the relationship view, the decision detail view is needed to judge if the context and the requirements behind the decision match.
- Logical decision groups in the relationship view are strong candidates for reusable paths.
- It is important to look at the goals of the decisions before reusing them.
- Decisions can be directly reused if the architectural goals behind the decisions are compatible with the goals of the new project.
- If the decisions' goals of the new project are not compatible with those of the documented project, then the decision views can still be used to identify candidate decisions that have to be reevaluated in other projects.
- Decision views provide reusable decision alternatives.
- Decision views allow reusing decisions for comparable problems.
- Decision views provide a basis for decision-making processes in other projects.
- By studying decision views for reuse, architects can make sure they do not forget decision options.

6.4.4 Interpretation

In this section, the results of the analysis are interpreted. At the end of the section, we discuss potential threats to the validity of this study.

Interpretation RQ1

Research question one is about the effort needed to create decision views according to our viewpoint specifications. In the analysis, we showed that the documentation of the decision views took approximately 0.65% of the total development effort of the software project under study. For the Sandnet project, this means that less than one hour (39 minutes) out of 100 person-hours had to be spent in order to create a relationship view, a chronology view and a decision detail view. The effort needed to create an individual view is hard to generalize, as it depends on the order in which the views were created. In this case study, the architects had not documented architecture decisions at all prior to the case study; thus all decisions had to be elicited in the first place. This took by far the greatest effort. Creating the detail view takes the second greatest effort. Finally, the relationship view and the chronology view follow in the effort ranking. Once decisions are documented in the detail view, the effort for creating the other views for the decisions is relatively low.

From the interviews with the architects, we learned that they would document the detail view first, but they would not describe every decision at the same level of detail. Some decisions might be more important or more complicated than other decisions; these decisions should be described in detail. Other decisions that are easy to comprehend do not have to be documented in detail. After the detail view, the architects would create a relationship view. The effort for creating the chronology view, in the opinion of the Sandnet architects, only pays off if projects are subject to long-term evolution.

It can be concluded that the effort for creating decision views is relatively low compared to the complete development effort. Depending on the available time, architects may choose not to document all views, but choose a subset depending on the characteristics of the concrete project. The architects mutually agreed that the effort for creating decision views is reasonable from the point of view of the project sponsors, which means the cost-benefit ratio of the decision views is low.

Interpretation RQ2

Research question two concerns the support for stakeholder communication offered by the views. The independent variables presented in the analysis show that the people who used the views to study the architecture of the Sandnet project were rather inexperienced with respect to architecture decision analysis. On average, they stated that they were rarely involved in the analysis of architecture decisions and that they have worked in the IT industry for less than ten years, which means that they are rather inexperienced technical stakeholders. On average, they took slightly more than one hour to study the decision views of the Sandnet project; none of them knew the project in advance. It is notable that the architects of the Sandnet project were astonished by the knowledge, the stakeholders gained about the project just by analyzing decision views. This impression is supported by the fact that the short preparation time was sufficient to identify major risks during the architecture review. The stakeholders concordantly stated that the relationship view is well suited to get an overview of the decisions made and to understand the relationships and dependencies between the decisions. The chronology view helped them to understand the evolution of the system and to get an insight into the architects' reasoning process. The decision detail view was seen as an important complement to the other two views in order to grasp the rationale behind a single decision. Finally, all stakeholders and the architects of the project agreed that decision views are a good means to communicate architecture decisions.

Interpretation RQ3

RQ3 is about the suitability of decision views to support architecture reviews. With one exception, the reviewers were rather inexperienced in performing architecture reviews. Nevertheless, the analysis showed that they were able to find 27 risks in the current architecture, from which the architects confirmed five risks as being important.

Decision views are beneficial for the preparation of architecture reviews, as they give

an overview over decisions made, show dependencies between decisions and allow comprehending the rationale behind single decisions. The chronology view allows reviewers to recap the evolution of the system and to quickly identify decisions that have changed since the last architectural review. The relationship view was found especially well suited for identifying important and critical decisions, performing impact analyses, and finding dependencies between decisions. The fact that rejected decisions and discarded decision alternatives are shown in the views allows the evaluation of single decisions quickly. Moreover, decision views support the architects in communicating the architecture to the reviewers in a structured way, thus ensuring that no important decisions are forgotten. The participants of the review mutually agreed that decision views are a good basis for architecture reviews that supports many review activities; particularly the introduction of the architecture to the stakeholders and the discussion of review scenarios.

Interpretation RQ4

Decisions views offer support for identifying reusable decision sub-graphs. Especially the relationship view was found helpful for identifying reusable decisions. The logical decision groups within this view are candidates for decision sets that can be reused as a whole. However, the reusability always depends on the context, in which decisions were made. Decisions from other projects can only be reused if the context and the architecturally significant requirements are comparable. The information from the decision detail view is essential to judge this for single decisions. One important aspect of decision views is that they record decision alternatives. This information is often not part of an architecture description, because many alternatives have not made it into the final architecture. Decision alternatives constitute valuable information for decision sub-graphs as they allow one to prepare decisions for question-option-criteria trees (MacLean et al. 1991), i.e., if requirement A then decision alternative B, if requirement C then decision alternative D and so on. Finally, studying decision views from projects in the same domain can help architects to make sure that no important decisions or decision alternatives have been forgotten.

Threats to validity

Construct validity is the degree to which the case is relevant with respect to the research questions (Runeson and Höst 2009). A frequent problem in case study research is that the case study design fails to clearly define operational measures, which allow one to objectively judge the collected data (Yin 2003). In this case study, we clearly defined the research questions prior to the data collection phase. The methods for the data collection were systematically selected in order to sufficiently address all four research questions. Furthermore, for every research question, we defined the dependent variables used as "measurements" to judge the data as well as the independent variables influencing those measurements.

We used different means to improve the internal validity of our findings. Internal validity concerns hidden factors, which affect the dependent variables (Wohlin et al. 2003). Using different types of triangulation can increase the reliability of the study results (Runeson and Höst 2009, Lethbridge et al. 2005). In this case study, we used several types of data source triangulation, e.g., by performing interviews with different people or by looking into different work artifacts. We also used methodological triangulation by combining different types of data collection methods. With the two types of triangulation, we made sure that every research question was addressed by more than one data source and by using different collection methods.

A potential threat to internal validity is the identification of the technical scenarios evaluated in the architecture review. The architects who defined the scenarios could have consciously or unconsciously defined scenarios that are well supported by the decision views. This could have resulted in a higher valuation of the decision views by the review team. This, however, was not the case. The scenarios were defined by the architects prior to and independent from the case study. They were documented as potential future changes or enhancements in the company wiki. To partially eliminate bias on the side of the architects, the review team chose three out of the five scenarios based on their own estimation of the scenarios' importance.

External validity is related to the generalizability of the results with respect to a specific population. External validity is regarded as a major problem in case study research because only one case is studied; which makes statistical generalization impossible (Yin 2003). We believe that our findings are valid at least for projects that have a comparable size (in person-months and development team) and use similar architecture approaches. However, external validity can only be shown (e.g., by analytic generalization (Yin 2003)) with at least a few replications. Although we have observed many of our findings in the three pilot projects (see Section 6.4.2) before conducting the case study, a systematic replication of the study in different projects and organizations is needed to support the claim for external validity.

6.5 Related work

Our work is related to the following fields within software architecture: architecture decision documentation and architecture decision views.

6.5.1 Decision documentation approaches

There are currently three main approaches to documenting architecture decisions. We briefly present these approaches and describe how our proposed viewpoints relate to each of them:

Decision Templates: Different templates have been proposed to describe architecture decisions in textual form, mainly using tables (see for instance (Tyree and Akerman 2005)). They can be used to capture relevant rationale behind decisions

including, among others, assumptions, alternative decision outcomes, the decision state, related requirements, and possibly related decisions. Tabular decision descriptions offer a certain degree of freedom for the decision documenters, because description elements can easily be added or left out. An additional benefit is their suitability for simple automated support such as through spreadsheets or wiki-type information systems, which offer out-of-the-box support for creating tables and linking textual elements to each other. No extra notations or special tools are needed to document decisions in textual form. However, decision tables tend to become very large and contain a lot of text. When many decisions are documented for one system, the overview of these decisions gets lost. Using templates, it is also challenging to visualize or trace complex relationships between decisions and to perform impact analyses. Especially for non-technical stakeholders and managers, long architecture documentation may seem daunting, which discourages them from reading the documentation.

The details captured using decision templates are of importance for framing a number of concerns. We have thus decided to include a viewpoint, the decision detail viewpoint, that uses a decision template similar to (Tyree and Akerman 2005). The views resulting from applying this viewpoint are especially important to record rationale for decisions. However, as mentioned before, they are not sufficient. Additional views are needed to provide an overview of the decisions made and to emphasize concerns that can hardly be satisfied in a table or catalog, e.g., decision relationships, decision chronology and the impact of decisions on stakeholders, on the architecture or on other decisions.

Annotations: Other approaches document architecture decisions using annotations (see for instance the Knowledge Architect Suite (Liang et al. 2009)). The respective tools allow users to attach "comments" to other architecture descriptions such as to UML or ADL models or to natural language text in text processing applications. They highlight elements as decisions and additionally capture relations, attributes, and the history of decisions. An advantage of using annotations is that architects and other stakeholders do not need not learn new tools to document decisions. Different stakeholders can typically use their preferred tools and attach annotations through specialized plugins to those tools. Furthermore, annotations are very well suited to elicit architecture decisions from existing project documentation. On the other hand, annotations from different tools must be combined with an additional decision management or documentation approach that allows to consume decisions without browsing multiple documents using multiple tools. Additionally, annotations do not provide instant support for the analysis of decision relationships, lifecycle management and consistency checks. This must be accomplished by additional tools collecting the annotated information and preparing them for further analysis (e.g., the Microsoft Word plugin used in (Liang et al. 2009)).

The elicited decisions from an annotation approach can serve as the raw data ba-

sis for creating decision views conforming to architecture decision viewpoints. Therefore annotation approaches are complementary to the proposed decisions viewpoints.

Decision Models: Decision models can present the same information as template-based and annotation-based approaches, but use dedicated models to represent decisions. Existing models within an architectural view are complemented with a decision model, which addresses decision-related concerns specific to the particular view (Kruchten et al. 2009, Duenas and Capilla 2005, Capilla et al. 2007). For example, in an architecture description following the 4+1 view model, a decision model in the deployment view would contain decisions about system deployment.

This approach provides a better overview of decisions than the aforementioned approaches and facilitates linking decisions to other architecture description elements. However, the problem remains that the existing approaches on decision models are dedicated to existing architectural viewpoints. The concerns addressed by the models are the same (or a subset of) concerns that are addressed by the viewpoint they are a part of; thus are system concerns. We, however, argue that beyond system concerns, architecture decision documentation must address additional concerns specific to architecture decisions. Nevertheless, architecture decision models can be complemented with our decision views if appropriate means for consistency among the decisions are taken. Additionally, the decisions from the decisions models can serve as a basis for further architecture decision elicitation.

6.5.2 Architecture decision views

The idea of introducing a dedicated decision view[3] to complement the traditional architectural views (e.g., the 4+1 view model (Kruchten 1995)) has been proposed by Kruchten, Capilla and Dueñas (Kruchten et al. 2009, Duenas and Capilla 2005). They emphasize the importance of documenting design rationale as a part of architecture decision documentation in architecture practice and identify a set of challenges and benefits of decision documentation. However, no concrete guidance is provided on how to define and construct decision views that integrate with view-based architecture documentation (ISO/IEC/IEEE 2011). The documentation framework presented in this chapter addresses many of the challenges identified in (Kruchten et al. 2009) and gives concrete advice on how to construct a set of consistent architecture decision views.

[3] An early draft of IEEE 1471 (version D1.0, dated February 1998) contained a decision viewpoint, described as: "The decision viewpoint documents the decisions about the selection of elements or characteristics. This viewpoint records the rationale for architectural choices. Typical models include: mission utility, cost/capability tradeoffs, element performance tradeoffs". However, all predefined viewpoints were removed from the standard before the final publication, leaving definition of viewpoints to its end users.

Various authors have proposed tools to visualize architectural design decisions (for a comparison of current toolsets, see (Shahin et al. 2010)). Graphical representations of decisions support stakeholders in understanding the architecture, as they allow them to visually inspect the architecture (Shahin et al. 2010, Lee and Kruchten 2008).

Some tools allow visualizing architecture decisions from different perspectives. A *perspective*, in this context, is a graphical representation of decisions suitable to address a set of decision-related concerns. While the idea of showing different perspectives of decisions has commonalities with the concept of multiple architecture decision viewpoints, it is not sufficient in isolation. A viewpoint is more than a perspective on architecture decisions, as it must provide concrete guidance on how to construct views, and it must ensure inter-model and inter-view consistency. Most importantly, it must integrate with other views used to document architecture. Our work is complementary to the tools analyzed in (Shahin et al. 2010); in fact, the creation of multiple views on decisions is only feasible in practice if appropriate tooling is provided to support the architects. This became evident in our case study. Identifying the best technique used to visualize the views presented in this chapter is, however, subject to further research. Candidate techniques should adhere to viewpoint definitions and be validated in industrial case studies.

6.6 Conclusions and future work

In this chapter, we introduced a documentation framework for architecture decisions consisting of four initial viewpoint definitions and the respective correspondence rules to ensure consistency among them. The four viewpoints, a Decision Detail viewpoint, a decision relationship viewpoint, a Decision Chronology viewpoint and a decision stakeholder involvement viewpoint, satisfy several stakeholder concerns related to architecture decision management. Furthermore, they can easily be integrated with other viewpoints to complete the picture of architectural design, decisions and rationale.

With the exception of the decision stakeholder involvement viewpoint, we validated the framework in an industrial case study and showed that the views can be created with reasonable effort. Furthermore, we showed that decision views facilitate communication between stakeholders, support technical architecture reviews and enable the reuse of architecture decisions. Although we could only find evidence for the suitability for *technical* architecture reviews in the case study, we believe that decision viewpoints are equally beneficial for non-technical architecture reviews and evaluations. Chapter 9 reports on our efforts to develop a decision-centric architecture evaluation method, which makes use of viewpoints from this framework.

The framework presented in this chapter comprises a coherent set of viewpoints that can be used as-is to document architecture decisions. However, our analysis of stakeholder concerns related to decision documentation (see Section 6.2) showed that some concerns cannot be satisfied optimally within the current set of viewpoints. In particular, concerns related to decision-requirements traceability and decision-design trace-

ability (C6, C7, C12) in Table 6.1 are currently under-represented and require additional research. To partially bridge this gap, we developed an additional viewpoint focusing on the representation of the relationships between architecture decisions and the forces that influenced the architect when making the decisions. The so-called *decision-forces viewpoint* is presented in Chapter 7.

Another direction for future research is applying the documentation framework for creating views in different orders and at different levels of detail. When applying our viewpoints in different projects and different organizational contexts, we observed that the viewpoints can be used in different ways. To give an example, in one project the decision detail view of architecture decisions was created on-the-fly during the decision making itself. This resulted in all decisions being documented with the same effort and the same information density. The decision relationship view was created after-the-fact at the end of the architecture phase. In another project, the relationship and chronology views were created on-the-fly during the architecting process, while only the most important decisions were thoroughly documented in the detail view afterwards. We plan to analyze the different ways of using the viewpoints in more industrial projects to come up with parameterized guidelines on how to construct the views in different organizational settings.

Finally, as mentioned before, effective tooling is vital for using viewpoints in the industry. We developed an open source web application (Open Decision Repository) to create views according to this viewpoint framework. Currently, the Decision Detail, Decision Relationship and Decision Chronology viewpoints are supported by the tool. The source code and documentation is located in a Google code repository and can be found under `http://opendecisionrepository.googlecode.com`. We are currently evaluating the Open Decision Repository in an industrial study as a pilot for larger-scale empirical validation.

Acknowledgement

We thank the following people for their valuable input and contribution: Stefan Arians, Dirk Bugzel, Mathias Deml, Christian Dietrich, Veli-Pekka Eloranta, Matthias Galster, Kai Koskimies, Christian Manteuffel, Dominique Petersen, Ben Ripkens, Christian Rossow, Sebastian Schmidt, Michael Stal, Martin Verspai, and the participants of the OPR software factory 2011.

Based on: U. van Heesch, P. Avgeriou, and R. Hilliard – *"Forces on Architecture Decisions – A Viewpoint"*, Proceedings of the Joint 10th Working IEEE/IFIP Conference on Software Architecture & 6th European Conference on Software Architecture, 2012.

Chapter 7

Forces on architecture decisions

Abstract

In this chapter, the notion of forces *as influences upon architecture decisions is defined. To facilitate the documentation of forces as a part of architecture descriptions, we specify a* decision forces viewpoint*, which extends our existing framework for architecture decisions, following the conventions of the international architecture description standard ISO/IEC/IEEE 42010. The applicability of the viewpoint was validated in three case studies, in which senior software engineering students used it to document decisions in software projects, two of which conducted for industrial customers. The results show that the forces viewpoint is a well-received documentation approach, satisfying stakeholder concerns related to traceability between decision forces and architecture decisions.*

7.1 Introduction

Decisions, and the rationale for those decisions, are pervasive elements of software architecture (Kruchten 2004a). Because of their crucial role, architecture decisions and rationale need to be captured and managed throughout the lifetime of a software architecture, as with any other important part of the architecture documentation. Moreover, decisions and their rationale should be documented in a form that integrates with the documentation of other types of architecture information in order to provide traceability between decisions and those other types.

Bosch proposed to capture the rationale behind an architecture using architecture decisions as first-class entities of architecture description (Bosch 2004). To date, different approaches have been presented to practically realize the documentation of architecture decisions; prominent among those are decision templates, as introduced by Tyree and Akerman (Tyree and Akerman 2005) (see Chapter 6 for a discussion of various decision documentation approaches).

ISO/IEC/IEEE 42010 (ISO/IEC/IEEE 2011) addresses the areas of recording architecture decisions and architecture rationale as part of an architecture description, specifying general requirements for decision documentation, but not particular mechanisms. As with any other kind of architecture information, architecture decisions and rationale pertain to different stakeholders' concerns. Consequently, a single form of representation is often not applicable to all concerns in a usable form; instead, different forms of representation, arranged as *architecture views*, can each effectively address a subset of concerns.

Since the earliest work on the foundations of software architecture by Perry and Wolf (Perry and Wolf 1992), and exemplified by Kruchten's 4+1 model (Kruchten 1995), the idea of documenting software architecture using multiple views has been widely adopted. IEEE Std 1471:2000 (IEEE 2000) first codified this practice of multiple views, with each view addressing specific concerns of interest to system stakeholders and introducing *viewpoints* to establish the conventions used in each view.

Building on this practice, in our previous work, we introduced a framework for architecture decisions using the conventions of ISO/IEC/IEEE 42010, containing an initial set of four viewpoints for architecture decisions: a decision detail viewpoint, a decision relationship viewpoint, a decision chronology viewpoint, and a decision stakeholder involvement viewpoint, each dedicated to specific decision-related concerns (van Heesch, Avgeriou and Hilliard 2012a) (an example of a decision-related concern is *What decisions are dependent on decision D?*)

In this chapter, we extend our earlier framework with the decision forces viewpoint (or shortly *forces viewpoint*), which is dedicated to establishing traceability between architecture decisions, stakeholder concerns, and the forces driving the decisions. Forces, in this context, include traditional requirements, but they also take the experience and expertise of the development team, as well as business and project constraints, into account. A force, in short, is a broad concept capturing anything that has a potential non-trivial impact of any kind on an architect when making decisions.

The forces viewpoint was validated in three case studies conducted with groups of senior students. Two of the groups worked independently on industrial software projects; the third group started an open source project as part of a module on Java EE. The results are promising, as they show that the forces viewpoint is well-received by the students, while satisfying many decision-related stakeholder concerns. Further, we learned that the forces viewpoint supports students in following a systematic and rational decision making process, when being created iteratively during the architecting process.

The rest of this chapter is organized as follows. Section 7.2 introduces the viewpoint framework and the basic ideas behind ISO/IEC/IEEE 42010. In Section 7.3, the decision forces viewpoint is specified. Section 7.4 reports on the case studies conducted to validate the viewpoint. In the next section, we briefly outline related work. Finally, in Section 7.6, we conclude and present areas for future work.

7.2 A framework for architecture decisions

In this section, the main ideas behind ISO/IEC/IEEE 42010 (ISO/IEC/IEEE 2011) and the framework for architecture decisions (van Heesch, Avgeriou and Hilliard 2012a), which were the basis for the development of the decision forces viewpoint, are briefly introduced.

7.2.1 ISO/IEC/IEEE 42010

ISO/IEC/IEEE 42010 is an international standard for the description of software architectures (and other kinds of system architectures). It is based on a few principles:

1. an *architecture description* (AD) expresses an architecture (of a system or other entity of interest);

2. an AD addresses the concerns of the system's stakeholders for that architecture.

3. the concerns drive the selection of the representation conventions (called *viewpoints*) used to express the architecture, each of which is dedicated to framing specific concerns;

4. consistency between the views is maintained using correspondence rules.

Building upon these principles, ISO/IEC/IEEE 42010 defines the required contents of individual ADs, the form of architecture description languages, and architecture frameworks.

7.2.2 Four viewpoints for architecture decisions

The framework for architecture decisions, introduced in Chapter 6, consists of an initial set of four viewpoints, each of which being dedicated to satisfying specific stakeholder concerns related to architecture decisions.

The **decision relationship viewpoint** makes relationships between architecture decisions explicit. Examples of decision relationships are *is caused by*, *depends on*, or *is alternative to*. Apart from relationships, views using this viewpoint document the current state of each decision in the system (e.g., *decided*, *approved*, or *rejected*). The **stakeholder involvement viewpoint** explains the responsibilities of specific stakeholders in the decision-making process. For example, views of this viewpoint show the stakeholders who proposed, confirmed, or validated particular decisions. The **decision chronology viewpoint** shows the evolution of architecture decisions over time. It also depicts architecture iterations and their endpoints (typically milestones, snapshots, or releases). The chronology viewpoint is the only viewpoint with a temporal component. All other types of views *freeze* a specific state of the architecture.

Whereas the previously mentioned viewpoints focus on specific aspects of architecture decisions to optimally frame their related concerns, the **decision detail viewpoint** is an aggregate viewpoint. This viewpoint combines the information shown in all other viewpoints, by giving detailed information about single architecture decisions. The detail viewpoint's model kind (a model kind establishes the conventions for all models in the respective view), at the same time acts as a shared metamodel for all viewpoints in the framework.

The decision forces viewpoint, introduced in this chapter, extends this existing set of viewpoints focusing on traceability between architecture decisions, stakeholder con-

cerns, and decision forces. In order to integrate the viewpoint into the decision framework, a few additions had to be made to the shared metamodel of all viewpoints. All changes are downwards compatible, which means that views created according to the viewpoints of the framework do not have to be changed after the integration of the forces viewpoint.

7.3 Decision forces viewpoint

Views using the decision forces viewpoint make explicit the relationships between architecture decisions and the forces that influenced the architect when making the decisions out of multiple alternatives. The term *force* is taken from the pattern community, which uses forces to elaborate on the description of a problem to be solved by a pattern's proposed solution. They define a force as "[...] any aspect of the problem that should be considered when solving it." (Buschmann et al. 1996). Similarly, when considering architecture decisions, a *force* is any aspect of an architectural problem arising in the system or its environment (operational, development, business, organizational, political, economic, legal, regulatory, ecological, social, etc.), to be considered when choosing among the available decision alternatives.

Forces arise from many sources; most often from requirements, but also from constraints, architecture principles and other "intentions" imposed upon the system; including personal preferences or experience of the architect(s) and the development team; and business goals such as quick-time-to-market, low price, or strategic orientations towards specific technologies (see (Mustapic et al. 2004) for an empirical study on influence factors on software architecture). Before making decisions, the architect assembles all forces relevant in the context of the system to be developed. It can be a good practice to maintain a list of typical domain-specific forces from different projects to make sure that not important forces are forgotten.

				View technology			Data storage	Middleware	DBMS	
				<decided>	<discarded>	<discarded>	<decided>	<discarded>	<discarded>	<decided>
				Java Swing	PHP	JSF	Central DS	EJB	MySQL	PostgreSQL
Decision Forces	**Architecture significant requirements**									
	Code	Description	Concern(s)							
	R1	Avg. response time <= 0.1s	Time behavior	++	+	-	-	-	+	+
	R5	Integrate mult. payment providers	Extendability	+		+		+		
	R6	Reliability of data storage	Reliability				++	+	+	++
	R8	Availability of full service (99.9%)	Reliability	++	+	+	+	-	+	+
	R9	Support growing no of users	Scalability	++	+	-	-	+	-	?
	R13	Security (personal data protection)	Security	+		?	+		?	?
	R16	Client platform independence	Portability	+	++	++				
	R23	Operability of user interface	Usability	++	+	+				
	R24	Communication via Internet	Network comm.		++	++	+	+	+	+
	R26	HBCI support	Banking protocols	+	?	+		+		
	R27	No licence costs	Development costs	+	+	+			++	++
	Other forces									
	F1	Inhouse experience	Development time							
	F1.1	Swing (very good)	Development time	++						
	F1.2	PHP (decent)	Development time		+					
	F1.3	JPA (good)	Development time							
	F1.4	MySQL (very good)	Development time				+		++	
	F1.5	JSF (very good)	Development time			+				
	F2	Strategic knowledge development	Competitiveness							
	F2.1	Learn Postgres	Competitiveness				+			++
	F2.2	Improve Javascript skills	Competitiveness	--	+	+				
	F2.3	Learn Jquery	Competitiveness	--	+	+				
	F4	Linux server available	Development costs		+	+	+	+		+
	F5	Non business criticality	Business criticality							+
	F7	Resource usage on server	Resource utilization	++	-	--	--	--	+	?

No resources needed. All calculations are performed on client side.

Figure 7.1: *Excerpt from a decision forces view (see 7.3.1 for conventions used here)*

Different forces may be orthogonal to one another, they may support, antagonize or contradict each other. Therefore, an architect must balance forces to make the best possible decisions. Figure 7.1 shows an extract from a decision forces view, which was created as part of a pilot study conducted to validate the design of the case studies reported below. In the pilot study, the decision viewpoints from the previously mentioned framework (Chapter 6) and the decision forces viewpoint were used to document architecture decisions made in a non-academic distributed open source online banking and accounting system for small and medium-sized companies.

The left part of the table shows the forces that were considered when choosing among the decision alternatives listed across the top of the table. Each force is classified by one or more concerns (please refer to Section 7.3.2 for an explanation of the relationship between forces and concerns). The decision alternatives can be grouped into decision topics (e.g. *view technology*, or *data storage* in Figure 7.1), if they were taken into consideration as alternatives to solve a particular problem. Within a decision topic, there can only be one decision with a state equal to or higher than *decided* (please refer to Chapter 6 for a description of all decision states). The comment box in Figure 7.1 contains an example of a textual description of a force-decision combination. The pluses and minuses indicate a positive or negative impact of a force on a decision alternative; an empty field means that a force is not applicable or neutral; a question mark expresses uncertainty. A more detailed description of the ratings can be found in Section 7.3.1. The

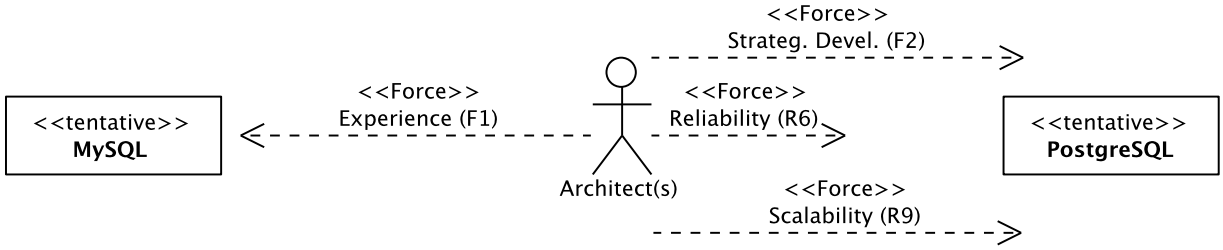


Figure 7.2: *Application of forces on an architect*

architect evaluates each architecture decision alternative in the context of the forces. As a result of the evaluation, a force can have a positive, negative, currently unknown, or neutral impact on the architect with respect to a decision; it either attracts the decision maker towards a specific decision alternative, or it repels the decision maker from an alternative, or it has no effect. Figure 7.2 illustrates the application of forces on an architect when choosing between two database management systems. On the one hand, the development team has a lot of experience using MySQL; this force attracts the architect towards choosing MySQL. On the other hand, the company wants to develop strategic knowledge with PostgreSQL, which is also more reliable than MySQL and turns out to scale better. In this particular case, after balancing these forces, the architect would probably choose PostgreSQL, provided that no other decision alternatives were taken into consideration. In a more general case, an architect would need to decide between

more than two options.

7.3.1 Forces viewpoint specification

Table 7.1 lists the decision-related concerns[1] framed by the decision forces viewpoint. These decision-related concerns are a subset of a larger set of concerns identified in Chapter 6. The original concerns C5, C6, and C7 referred to requirements rather than forces. They were renamed to acknowledge that decisions should be linked to all types of forces, instead of linking them to requirements only. The codes in Table 7.1 were copied from Chapter 6 for consistency.

Views of the decision forces viewpoint are dedicated to supporting decision–force traceability. They can be used by stakeholders interested in decision rationale, decisions relevant for specific stakeholder concerns, addressed requirements, conflicting forces and how these all relate to each other. The main stakeholders for this viewpoint are architects, but also reviewers and other stakeholders who need to comprehend the choices made in the architecture. Table 7.2 shows the stakeholders along with their main decision-related concerns with respect to the forces viewpoint. Similarly to the decision-related concerns, the stakeholders were identified in our previous work.

Table 7.1: *Concerns of the decision forces viewpoint*

Code	Concern
C3	What is the rationale for decision D?
C4	What concerns C_i does decision D pertain to?
C5	What forces F_j impact/influence decision D?
C6	What decisions D_k are influenced by force F?
C7	What forces F_l have conflicting influences on decision D?
C23	What decisions D_p or decision sub-graphs SG_q can be reused in other projects?

The decision forces viewpoint consists of a single model kind. Figure 7.3 depicts its metamodel, which presents the conceptual elements for architecture models that adhere to it. This model is part of a shared metamodel, which is used by all viewpoints of the decision documentation framework. Together with well-defined correspondence rules, the shared metamodel ensures consistency among the views of different viewpoints. The elements in Figure 7.3 with a gray background map to the corresponding elements in Figures 2 and 4 of ISO/IEC/IEEE 42010. In the following, each of the elements used in Figure 7.3 is briefly described.

An architecture decision pertains to one or more concerns. Forces views show only the current state of each decision (e.g. *decided*, or *discarded* Chapter 6). While decisions

[1]The term *decision-related* concern is used to refer to concerns pertaining to decision documentation (as opposed to any other types of stakeholder concerns which are simply termed *concerns*).

Table 7.2: *Typical stakeholders of the decision forces viewpoint and their concerns*

Stakeholder	Concerns
Architect	C3, C4, C5, C6, C7
Reviewer	C3, C4, C5, C6, C7
Requirements Engineer	C4, C6, C7
New project member	C3
Domain expert	C23

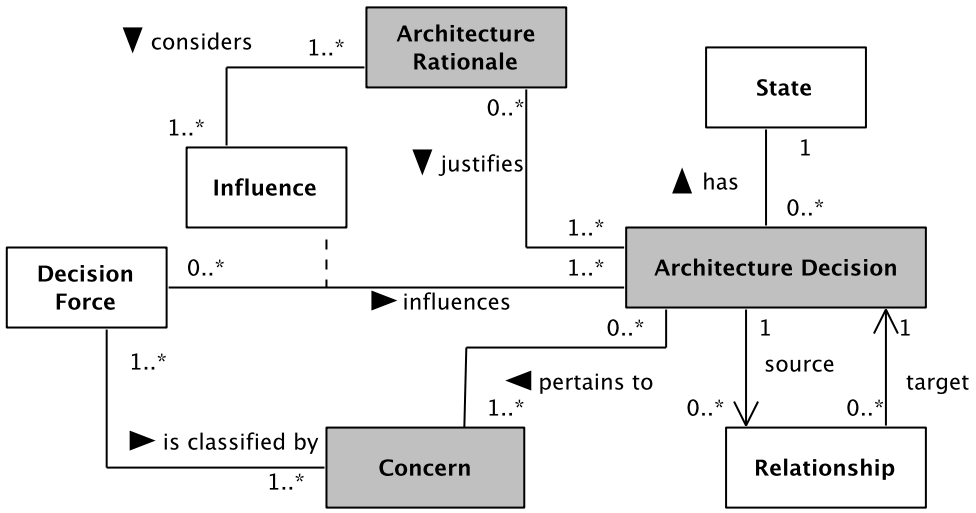


Figure 7.3: *Metamodel of decision forces viewpoint*

can generally have different types of relationships with each other, the forces viewpoint only regards the *is alternative for*-relationship to group multiple decision alternatives into a decision topic.

According to ISO/IEC/IEEE 42010, "Architecture rationale captures explanation, justification or reasoning about architecture decisions that have been made." (ISO/IEC/IEEE 2011). In terms of the forces viewpoint, the architecture rationale should balance all relevant forces that influence a decision. Note that architecture rationale is not described in forces views; it is documented explicitly in decision detail views, which are part of the decision framework. In the forces viewpoint's model kind, the association between *Architecture Rationale* and *Influence* implies that the rationale description should consider the relevant forces.

All forces are classified by one or more concerns. A stakeholder could for instance be concerned about development cost, while concrete forces classified by this concern could be "not to use paid 3rd-party licenses", or to "use available hardware where possible". The force not to use 3rd-party licenses could, besides the development cost con-

cern, be classified by a legal concern (e.g how the software can be distributed).

Apart from a textual qualification, the *influences* relationship between decision force and architecture decision can take one of the following values, estimated by the architect(s) of the system:

++: A force strongly supports a specific decision alternative to be chosen. An example from Figure 7.1 is the operability force, which strongly advocates the choice of Swing/Java, because Swing can be used to develop rich graphical user interfaces.

+: A force moderately supports an alternative.

blank: A force has a neutral influence on a decision alternative, or it is not applicable.

-: A force moderately opposes an alternative.

- -: A force strongly opposes an alternative to be chosen. For instance, if the programming team has no experience in functional programming, then this would be a strong argument against choosing Lisp or Haskell as a programming language.

X: A decision alternative is prevented by a force. For instance, a force could be not to use libraries distributed under an open source license. Such a force would for instance prevent the use of Apache Lucene as a search library. Nevertheless, it can make sense to document such a decision alternative, because the forces view could be used to negotiate constraints or requirements with the customer, if its advantages clearly outweigh the opposing forces.

?: It is currently unclear how the decision alternative is impacted by a force. This rating should be temporary, indicating that prototyping, or more research has to be done to understand the impact better.

Constraints and cross-viewpoint correspondence rules relevant to this viewpoint are specified in Appendix C.

7.3.2 Stakeholder concerns versus decision forces

In the context of ISO/IEC/IEEE 42010, the term *concern* was chosen to include any interest that stakeholders consider fundamental to the architecture of the system (including the process of creating the architecture): "Concerns arise throughout the life cycle from system needs and requirements, from design choices and from implementation and operating considerations." (ISO/IEC/IEEE 2011). The standard introduces stakeholders' concerns as a means to drive the selection of architecture viewpoints, i.e. different stakeholders for the architecture description have different needs in terms of different kinds of information. Therefore, concerns result in selecting appropriate representations of the architecture. Forces, in contrast, do not drive representational choices but architecture decisions. The concept of a force is related to the concept of a concern, in that all forces are classified by concerns (see Figure 7.3). If a force could not be classified by

at least one concern, this means that it would not represent any interest of the relevant stakeholders.

7.4 Three case studies

To validate the usage of the decision forces viewpoint in software projects, we conducted a multiple-case study with senior students working on non-academic software projects. A case study was preferable over surveys or experiments, because the phenomenon (i.e. the influence of the forces view documentation) had to be studied over a long period of time, thus limiting the possibility for strict control of independent variables (Runeson and Höst 2009). Additionally, a multiple-case design is regarded as more robust than single-case studies, because conclusions from one case can be compared to other cases (Yin 2003), which increases external validity.

7.4.1 Study goal and research questions

Following Robson's classification scheme (Robson 2011), this multiple-case study is exploratory in nature. The goal is to explore the support provided by the decision forces viewpoint to software architecture activities and the coverage of decision-related concerns in software projects. In particular, the study aims at answering the following two research questions:

RQ1: How does the forces viewpoint support the decision making process?

RQ2: Which of the decision-related concerns mentioned in Table 7.1 does the forces viewpoint support?

7.4.2 Study design and execution

Case descriptions

The study was conducted in the context of two lecturing modules in the software engineering study program at the Fontys University of Applied Sciences in Venlo, the Netherlands. In total, we observed three student groups working on different projects. Two of the projects were conducted as part of a lecturing module, in which student groups work on tasks for external, industrial customers[2]. The third project was done as part of a lecturing module on the Java enterprise edition (JEE). In this module, the students were free to make up their own software project, as long as it involved at least one technology from the JEE specification set. In all cases, the students worked on their own responsibility without lecturers intervening in their decision making process. The decision documentation was no integral part of the modules and was not graded. One of the authors was involved in the third case as a lecturer, while none of the authors was

[2]The customers have asked us to stay anonymous.

involved in the former two cases. All projects were observed over a period of seven weeks. In the following, the three projects are briefly described:

PrjA: This project is a further development of a legacy documentation system used to generate different types of documents based on templates and dynamically allocated data. The software project was commissioned by a medium-sized German software company. A prominent user of the system is the Bavarian Department of Justice. The primary task of the project group was an architectural re-design to a service oriented architecture, including the migration of the existing functionality to services and the choice and usage of an appropriate enterprise service bus technology.

PrjB: This software was ordered by a Dutch company that acts as a broker between restaurant owners and cooking personnel, specialized on catering, cooking workshops, and interim executive chefs. The student group had to develop a web application for personnel services in the gastronomy business, allowing freelancing cooks to register and apply for jobs. Job offers can be posted by restaurant owners, for instance. The software had to be developed from scratch.

PrjC: The third project was conducted as part of a lecturing module on JEE. The students in this group started an open source project called */notes* (pronounced Slashnotes) for managing, sharing and distributing notes. The software offers three different clients that can be used to access notes: a web application based on JQuery, a Java desktop application (using Swing), and a mobile client for Google's Android operating system. All architecture decisions had to be made by the students. A short video showing the main features of the application can be found on YouTube (`http://youtu.be/wW1Lgq2gZvg`).

Subjects

The subjects of the study were students from the last year of a four-year software engineering program of study. All of the students had already gained some industrial experience from a five-month internship; some of them had additionally pursued part-time jobs in the software engineering industry. During the course of their study, the students had followed different courses on programming, object-oriented analysis and design, and software engineering process models (e.g. RUP, Scrum, Iterative waterfall). To gather their experience regarding programming, design, and software architecture; as well as the time they had already spent in the industry, we asked all participants to fill in a web-based questionnaire prior to the study. Table 7.3 shows the number of students in each group (No. stud.), as well as the average number of months of experience that the students had as programmers (Prog. exp.), as software designers (Des. exp.), with software architecture (Arch. exp); and as software engineers in the industry, or as payed freelancers. The numbers in parentheses show the standard deviations. With the exception of one outlier in PrjA regarding programming, design, and industrial experience,

Table 7.3: *Previous experience of the subjects*

	Project A	**Project jB**	**Project C**
No. students	6	5	4
Programming experience	75.33 (48.89)	49.2 (13.26)	59.5 (21.56)
Design experience	50.33 (28.63)	32.2 (3.03)	33.5 (8.54)
Architecture experience	38.67 (4.84)	28.6 (9.48)	11.25 (7.97)
Industry experience	25.17 (36.21)	7 (2.83)	7.25 (6.18)

the students' previous experiences was comparable between the groups, which renders them equivalent data sources. The fact that the students were in the last semester before the graduation project, and had some first experiences in the IT industry, makes them suitable subjects for the population of inexperienced software engineers at the beginning of their professional careers.

Carver et al. provide a checklist for conducting empirical studies with students (Carver et al. 2010). This checklist was used to ensure that the study had a pedagogical value for the participating students and that the results are generalizable to a larger population (in this case the population of inexperienced software engineers). In the following, we list all items of this checklist together with a brief explanation on how the checklist item was considered:

1. **Ensure adequate integration of the study into the course topics** – In both lecturing modules, the students had to make architecture decisions autonomously. The decision forces view supports the decision making process and provides decision-force traceability. Thus, it integrated well into the course topics.

2. **Integrate the study timeline with the course schedule** – The timeline for the study was explicitly planned according to the start of the lecturing modules.

3. **Reuse artifacts and tools where appropriate** – The students used a spreadsheet application for creating the decision forces view. No special tool was introduced for the purpose of decision documentation.

4. **Write up a protocol and have it reviewed** – A study protocol was written before the study and reviewed by the authors in multiple iterations.

5. **Obtain subjects' permission for their participation in the study** – Prior to the two courses, the students were asked if they wanted to participate in the study. They were ensured that no personal data would be made available in the study report. All students expressed their interest in the study. They were also given the opportunity to withdraw from the study by sending an email to the course lecturers.

6. **Set subject expectations** – The students were informed about the effort, we estimated for the decision documentation. Apart from that, we told them that we would give them feedback about how to improve their individual architecting processes after the study.
7. **Document information about the experimental context in detail** – The context of the study is documented in this article.
8. **Implement policies for controlling/monitoring the experimental variables** – The relevant previous experience of the subjects, as well as the descriptions of the projects they were involved in, are reported in this chapter. The data collection methods and data sources used to monitor these variables are described in Section 7.4.2.
9. **Plan follow-up activities** – At the end of the semester, the students were informed about the study results. Each project group also received individual feedback on their architecting process.
10. **Build or update a lab package** All collected data was stored in a digital study database (as proposed in (Yin 2003)). The database was used as a basis for the analysis.

Data collection

The data collected in this case study is qualitative in nature. We applied triangulation of data-sources, which is a well-accepted method to increase the precision of studies that mainly collect qualitative data (Runeson and Höst 2009, Yin 2003, Stake 1995). The different data sources that were triangulated, correspond to different data collection methods, which are as follows:

Work artifacts: In the two lecturing modules, from which we recruited our project groups, the students were obliged to store all project related files in Subversion repositories. The researchers were given read access to these repositories, enabling them to track the progress and the iterative refinement of the architectural design.

Focus groups: At the end of the seven weeks, we conducted focus groups with each of the projects. Focus groups are group interviews with a small number of participants, in which a moderator asks questions to concentrate the discussion on a predefined topic. In contrast to individual interviews, focus groups allow group members to build up on each others' answers leading to more profound information (Kontio et al. 2008). All focus groups were audio recorded and transcribed.

Participant observation: During the seven weeks, the three groups were regularly, at least weekly, visited during their working sessions. The researchers took field notes about their observations, which were afterwards scanned and stored in the study database.

Pilot study

To fine-tune the design of the study, in particular the data collection procedures and the research questions, we performed a pilot study. In this pilot study, we used the decision framework, and the forces viewpoint in particular, to document the architecture decisions of a system for online banking and accounting. One of the authors was involved in the project as a developer. Figure 7.1 shows an excerpt from the forces view created in this pilot.

The pilot study was particularly helpful for understanding how the forces viewpoint can support the decision making process. In addition, the results were used to develop the question guide, which was employed during the focus groups to ensure that no important topics of interest were forgotten.

7.4.3 Analysis procedure and results

As the data in our study database was qualitative to a large extend, we chose to apply a grounded theory approach (Glaser and Strauss 1967) to analyze the data. While being used mainly in social sciences, grounded theory has recently also gained more attention in software engineering related research (see for instance (Adolph et al. 2011, Urquhart et al. 2010)).

Analysis procedure

Grounded theory is inherently explorative in nature, as it promotes the analysis of data without predetermined ideas about potential findings. Concepts emerge slowly by constantly comparing indicators found in the data to previously identified indicators. That way, an idea about a finding (usually referred to as a theory) is either supported by additional evidence, or it has to be rejected if no additional indicators can be found to carry it. In the following, the steps we followed during the data analysis are briefly described. Note that steps two to four are performed iteratively.

1. **Convert data to PDF**: The gathered data was exclusively stored digitally. As a preparation for the data analysis, we converted all files in the study database to the PDF format to allow for a uniform coding procedure.

2. **Coding**: All PDFs were intensively studied. Indicators for concepts related to decision views (in particular the forces view) were coded (i.e. labelled) as brief statements using PDF annotations. Please refer to Adolph et al. (Adolph et al. 2011) for an extensive explanation of the terms *indicator*, *code*, *concept*, and *category*, which are central concepts in grounded theory.

3. **Identify concepts**: During the coding procedure, concepts emerge, which represent candidate patterns of behavior, suggested by a set of indicators. The concepts

were registered and related to the codes supporting it. The result after some iterations of analysis, was a set of concepts describing how the three student groups used and perceived the forces viewpoint in their projects.

4. **Classify concepts into categories**: Finally, in the last step of the analysis, the concepts from the three groups were compared to identify common categories of concepts. A category is a concept on a higher level of abstraction. As stated above, findings that were concordantly made in more than one project group are more reliable.

Analysis and interpretation

Table 7.4: Result of the qualitative analysis

Code	Category	PrjA	PrjB	PrjC	Conc.	RQ
Cat1	Required students to think more carefully about decisions.	X	X	X		RQ1
Cat2	Triggered students to consider quality attribute requirements.	X	X	X		RQ1
Cat3	Prevents ad-hoc decisions.	X	X	X		RQ1
Cat4	Forces viewpoint will be used in other projects.	X	X	X		RQ1
Cat5	Triggered students to identify more alternatives.	X	X			RQ1
Cat6	Good way to document decisions.		X	X		RQ1
Cat7	Creating the forces view took a lot of time.	X				RQ1
Cat8	Prevents inefficient discussions about decisions.	X				RQ1
Cat9	Created with reasonable effort.	X				RQ1
Cat10	Saved time in the end.		X			RQ1
Cat11	Support for rational decisions.			X		RQ1
Cat12	Forces view complements relationship view.			X		RQ1
Cat13	Useful for architects, designers, programmers, and new project members.			X		RQ1
Cat14	Support for weighing forces is missing.			X		RQ1
Cat15	Identifying all forces is a matter of experience.			X		RQ1
Cat16	Forces view and relationship view are simultaneously refined.			X		RQ1

Table 7.4 – continued from previous page

Code	Category	PrjA	PrjB	PrjC	Conc.	RQ
Cat17	Proper tool support needed.			X		RQ1
Cat18	Maintain overview over architecture decisions, concerns, and forces.	X	X	X	C4, C5, C6	RQ1, RQ2
Cat19	Helpful to systematically compare decision alternatives in the context of forces.	X	X	X	C5, C6	RQ1, RQ2
Cat20	Help for estimating requirements coverage.	X		X	C6	RQ1, RQ2
Cat21	Support for systematic trade-offs between forces.			X	C7	RQ1, RQ2
Cat22	Supports sharing architecture rationale.	X	X	X	C3, C23	RQ2

Table 7.4 summarizes the results of the qualitative analysis. The table maps the categories, identified in step 4 of the analysis procedure, to the project groups, in which they were observed. Additionally, the table shows decision-related concerns (column *Conc.*) that are related to some of the categories, as well as research questions (column *RQ*), to which the categories contribute. In the following, the results are interpreted in the context of the two research questions. The interpretation focuses on categories that were recognized in at least two of the projects; only regarding suggestions for improvement, we discuss categories assigned to single groups only.

RQ1: How does the forces viewpoint support the decision making process? As Table 7.4 shows, the data collected from all three groups indicated that the forces views caused the students to take the decision making process more seriously than they would have done otherwise (**Cat1**). The fact that decisions and forces had to be documented explicitly caused the students to think more concretely about available decision alternatives (**Cat5**), and the forces that influence the choice between these alternatives. The students noticed that the view prevented them from making decisions ad-hoc (**Cat3**, **Cat19**). A comment in a focus group was "If you don't have the view, then you might also see alternatives, but if I have experience in a solution then I will choose this one. But with the (forces) view, you are forced to think about which one is really better." It is notable that all groups mentioned that the forces views triggered them to consider quality attribute requirements in the first place (**Cat2**). They had not thought of this in projects before (during their studies or in side jobs). Among all collected work artifacts, the forces views were the only documents in which quality attributes were mentioned. Considering quality attributes in architectural design, however, is an important best-practice that should be adopted by inexperienced software engineers (see Chapter 4).

In general, the forces viewpoint was very well received by the students. They found it especially helpful to maintain an overview over decisions made and the factors that influence the decisions (**Cat18**). The majority of members in all groups explicitly stated

that they will reuse the forces viewpoint in future projects (**Cat4**)[3]. They acknowledged that it is a good way of documenting architecture decisions (**Cat6**). This finding is particularly important, because our experience from multiple studies with students shows that they cannot be convinced to document their decisions using decision templates (e.g. from (Tyree and Akerman 2005)). They usually perceive decision documentation as a tedious task that does not have an immediate benefit. The forces viewpoint, in contrast, is a documentation approach that they quickly accepted; presumably because of its relative light-weightiness and its immediate support for the decision making process.

Although the students were predominantly positive about the forces viewpoint, they also made suggestions for improvement. ProjectC was concerned about the fact that the forces viewpoint does not provide means to specify different weights for forces (**Cat14**). In their project, some forces were clearly more important than other forces causing them to select an architecture decision alternative that had a lower rating (i.e. sums of pluses and minuses) than the other alternatives. Although we had considered this aspect during the design of the forces viewpoint, we chose not to include it in the viewpoint specification to keep it simple. Systematically weighing forces would have introduced additional complexity, which could have deterred students from using the view properly. However, the forces viewpoint can easily be customized by stakeholders in order to introduce such weights in their projects. Apart from this, it became evident that identifying all relevant forces is a matter of experience (**Cat15**). Therefore, especially for domain-specific forces, it can be helpful to collect typical forces from different projects that can be used as a checklist to ensure that no important forces are forgotten. Tool support would also be appreciated, especially to ensure consistency and to save work when creating the forces view in addition to other views from the framework (**Cat17**).

RQ2: Which decision-related concerns does the forces viewpoint support? To find out for which decision-related concerns the students used the forces views, we analyzed the concepts and categories and compared them to the list of concerns in Table 7.1. The results are shown in Table 7.4 (categories 18 to 22). Because the categories are conceptually more abstract than single concerns, sometimes multiple concerns are mapped to a single category. Note that the students were not knowledgable about the concerns we had assigned to the forces viewpoint in the specification. This would have introduced a threat to the validity of our findings.

The concepts classified under category **Cat18** have shown that all three groups used the forces views to maintain an overview over architecture decisions, concerns, and forces. The students described that one column in the forces view (see Figure 7.1) shows which concerns (**Cat18**, concern C4), and which forces (**Cat18**, concern C5) are related to a decision. They also understood that a row in the view shows decisions influenced by a specific force (**Cat18**, concern C6). This information was actively used by the students to make the choice between multiple alternatives more systematic (**Cat19**, concerns C5, C6).

[3]At the time this paper was written, the students were working on their final bachelor projects in external companies. We repeatedly received questions and suggestions about the forces viewpoint, which indicates that at least some students indeed keep using decision views.

All three groups saw value in the forces viewpoint with respect to sharing architecture rationale (**Cat22**, concern C3). In particular, they mentioned that usually individual members of the groups were more knowledgeable about specific architecture decision alternatives and their relation to forces than others. The forces views helped them to spread this knowledge better among the group members. Using their own words, the student groups stated that studying the forces view helped everybody to understand the why behind architecture decisions, including the decisions primarily made by others. Category **Cat22** was also assigned to concern C23, because the students saw the potential of the forces views to facilitate the reusability of decisions in other projects: by providing the rationale in terms of decisions addressing specific forces, the decisions can be reused in cases where similar rationale would make sense.

Two groups used the forces views to estimate the coverage of some important requirements (**Cat20**). During the analysis of the work artifacts, we could see that all groups had used requirements as forces; only two of the groups, however, had also actively used the forces view to check in how far the decisions made were suitable to actually satisfy the requirements. They understood that a row in the view shows all decisions that need to be regarded when estimating the coverage of a particular requirement (i.e. a force in the forces view). For the same reasons, **Cat20** confirms concern C6, which is about identifying all decisions that were influenced by a particular force.

Concern C7 (Which forces have conflicting influences on a decision?) was only explicitly approved by one project. Conflicting influences have to be regarded when making trade-offs (**Cat21**). In forces views, conflicting impacts are indicated by a decision that has positive rating for one force and negative ratings for another force. Although this situation was observed in the forces views of all three groups, only one of the groups explicitly acknowledged the usefulness of forces views for making trade-offs. We conjecture that the other groups did not mention trade-offs, because they had not explicitly discussed such situations. Only in PrjC, we observed that the group actively and fully-aware discussed conflicting impacts and ways to compensate resulting issues. This corresponds to the team's earlier discussed statement that they were missing weights for forces (**Cat14**). Particularly when making trade-offs, different weights of forces should be considered.

7.4.4 Threats to validity

In the following, we present potential threats to the validity of our findings. In particular, we cover typical validity threats in software engineering studies, as identified in (Yin 2003) and (Wohlin et al. 2012).

Construct validity

Construct validity is concerned with the operational measures taken to analyze the phenomenon under study. In this case, we used multiple sources of evidence (i.e. work artifacts, field observation, and focus groups) to study the use of the forces viewpoint

in software projects. Additionally, the use of a grounded theory approach ensures that conclusions are rooted in the collected data and that no important concepts are forgotten.

Internal validity

Internal validity mainly has to be considered in explanatory case studies (Yin 2003), in which a cause-effect relationship is going to be established. In exploratory case studies, internal validity basically concerns making inferences. In this case, we tried to address this potential threat by involving different sources of data, including direct participant observation and analysis of work artifacts. Logical deductions are generally based on multiple sources of evidence and aligned among at least two of the projects under study (we did not make deductions from data coming from only one project).

External validity

External validity concerns the generalizability of the study's findings to a larger population. Because statistically representative samples can typically not be achieved in cases studies, the emphasis is usually put on analytical generalization, thus an explanation why the findings are representative for other cases with common characteristics (Runeson and Höst 2009). Yin points out that external validity can be improved by using replicated study-designs (Yin 2003). In this study report, we present results that are based on findings made in three different cases using identical study designs. This reduces the influence of the concrete cases and of the individual students in the different project groups. Therefore, we assume that our findings are relevant at least for the population of inexperienced software engineers at the beginning of their professional careers. Although we did not find any indicators raising legitimate doubts about the usefulness of the decision forces viewpoint for experienced software architects as well, additional industrial studies must be conducted to generalize the study results to this larger population.

Reliability

The reliability of a study is concerned with the minimization of errors and biases that stem from the researchers who conducted the study. In this case, the moderator of the focus groups could have influenced the students towards giving specific answers. This threat was mitigated by asking open questions like "How did the decision forces view influence your decision making process?". As follow-up questions, the moderator asked the students to explain their answers, or to go more into detail. To mitigate the risk of suggestive questions and to make sure that all important topics would be covered, we prepared a question guide (as described by Mack et al. (Mack et al. 2005)) in advance, which was used by the moderator during the focus groups.

An additional potential threat to reliability could result from students not staying true to the facts during the focus groups. To mitigate this risk, we used data-source tri-

angulation (Lethbridge et al. 2005), which allowed us to verify concepts using different types of data. Additionally, as stated above, we prioritize results that were concordantly found in at least two of the three case studies.

7.5 Related work

The work presented in this chapter is related to architecture decision documentation in general, and architecture decision views in particular. In Chapter 6, we extensively discussed related work in these two fields. Therefore, in the remainder of this section, we focus on related work with respect to traceability between requirements (problems) and design (solutions).

The decision forces viewpoint acknowledges the importance of relating architecture decisions to the forces driving those decisions. As such, the forces viewpoint is connected to the research area of relating architecture and rationale. In their recent book, Avgeriou et al. compiled 15 articles that relate architecture and requirements (Avgeriou et al. 2011), taking among others traceability between architecture design, decision rational and requirements into account. Tang et al. in the same publication, provide a traceability metamodel for bridging the gap between elements from the problem space (stakeholders, requirements, and issues) and elements from the solution space (architectural design, structure, components) using architecture decisions and rationale as intermediaries. Other authors had proposed to use reference models to support different types of requirements traceability before (e.g. (Gotel and Finkelstein 1994, Ramesh and Jarke 2001)).

A slightly different approach to software architecture–requirements traceability has recently been introduced by Malavolta et al. (Malavolta et al. 2011). Originating from the model-driven architecture field, they suggest to use weaving models to relate requirements models, architecture decision models, and different types of architecture descriptions. In contrast to using one shared metamodel, weaving models are non-invasive and provide greater flexibility.

The conceptual elements used in the forces viewpoint have some similarities with the elements of design space analysis (MacLean et al. 1991). MacLean et al. propose to represent design rationale as questions, options, and criteria. A decision topic, as used in the forces viewpoint, could be formulated as a question, e.g. *which database management system to use?*, the options are similar to the decision alternatives and criteria could be expressed as forces. However, there are also substantial differences between design space analysis and the forces viewpoint. First of all, the concept of a force is broader than the criterion concept in design space analysis. MacLean et al. describe that criteria "represent the desirable properties of the artifact (i.e. an element in the design space) and requirements that it must satisfy". Apart from properties and satisfied requirements, forces include any type of contextual factors, e.g. other decisions made, experience of the development team, or politics. Another difference lies in the assessment of the forces, or criteria respectively. The design space analysis approach only distinguishes

between a positive assessment and a negative assessment, whereas the forces viewpoint allows for more fine grained evaluation and a qualitative description of this evaluation. As a consequence of the mentioned differences, each design space analysis diagram can be represented as a forces view, but not the other way round. The forces viewpoint also differs from design space analysis in the form of the representation. Forces views are tables that allow to efficiently trace forces, decisions, and impact ratings; design space analysis relies on the usage of tree structures. One of the downsides of a tree representation is that it can hardly be used to identify which design questions were impacted by a specific criterion, because the same criterion can appear multiple times as a leaf in the tree. As opposed to the forces viewpoint, design space analysis does not consider the case that a design option is taken into consideration for multiple decision topics, or questions respectively.

Tang et al. provide an architecture model for design traceability and reasoning (Tang et al. 2007). The model connects architecture description elements (as defined in IEEE Std 1471-2000 (IEEE 2000)) to architecture decisions and architecture rationale, as first class entities. These authors also implicitly acknowledge the existence of decision forces, by introducing a concept they call *motivational reason*. A motivational reason can be among others a requirement, a goal, an assumption, or a constraint.

Despite this existing work on architecture rationale-design traceability, to the best of our knowledge, no approach exists that systematically integrates this traceability in a software architecture description following the conventions of ISO/IEC/IEEE 42010. Additionally, and more importantly, very few authors have recognized the importance of treating the full scope of decision forces extending across the context of the system and the environment in which it is developed, as first-class entities in an architecture description. We argue that the concept of decision forces, as introduced here, is a valuable contribution to the field.

7.6 Conclusions and future work

In this chapter, we introduced the decision forces viewpoint as an extension to our framework for documenting architecture decisions. The viewpoint was validated in a multiple-case study, which has shown that the forces viewpoint is very well received, while satisfying its related concerns. Additionally, the forces viewpoint has demonstrated its ability to support inexperienced software engineers during the decision making process, by providing a structure that triggers them to consider multiple architecture decision alternatives and systematically compare them in the context of all important forces.

We are currently observing the use of the forces viewpoint and other decision viewpoints from our framework in an industrial study, in which we analyze the suitability of decision views for problem and design space documentation. Apart from that, we have used it as part of the decision-centric architecture evaluation method, presented in Chapter 9.

Finally, as suggested by many users of our decision viewpoints, we continue the development of a tool suite, which efficiently supports architects in documenting views corresponding to our viewpoints.

Acknowledgements

We would like to thank all participating students from the software factories and the Java enterprise edition course 2011/2012. Two of the cases reported on in this chapter are part of a larger study designed and conducted together with Antony Tang.

We would also like to thank Veli-Pekka Eloranta and Kai Koskimies, with whom we initially discussed the concept of decision forces in the context of decision-centric architecture evaluation.

Part IV

Supporting architecture decisions

Submitted to the Journal of Systems and Software as: U. van Heesch, P. Avgeriou, A. Tang – *"Does decision documentation lead to more rational decisions? - A comparative multiple-case study'*

Chapter 8

How decision documentation affects the reasoning process

Abstract

Software architecture design is challenging, especially for junior software designers. Lacking practice and experience, junior designers need process support in order to make rational architecture decisions. In this chapter, we present the results of a comparative multiple-case study conducted to find out if the decision viewpoints presented in Chapters 6 and 7 can provide such a support. The case study was conducted with four teams of software engineering students working in industrial software projects. Two of the four teams were instructed to document their decisions using decision viewpoints; the other two teams were not instructed to do so. We observed the students for a period of seven weeks by conducting weekly focus groups and by analyzing their work artifacts and minutes. Our findings suggest that junior designers who use decision viewpoints are more systematic in exploring and evaluating solution options. However, the decision viewpoints did not help them in managing requirements and complexity.

8.1 Motivation and background

Software architecture design is a demanding task which requires designers to find optimal solutions within a specified timeframe for often vaguely defined requirements, while managing risks, regarding constraints, and taking business drivers into account. There is a steep learning curve to becoming a good architect: junior software designers usually have to endure extensive periods of learning, going through numerous painful trial and error attempts when making architecture decisions.

Before becoming software architects, junior software designers need to develop a) a certain body of knowledge, and b) the cognitive skills for systematically reasoning about architecture decisions. These two factors are important for making rational decisions. A rational decision is a decision based on the application of reason. A rational decision deliberates the benefits and drawbacks of the available design options, while taking requirements and other project constraints into account.

Junior software designers need guidance to handle the inherent complexity of rational decision making, especially with software architecture issues. Explicitly modeling architecture decisions during the design process may provide such a guidance. Although being widely overlooked in software engineering education, the treatment of architecture decisions as first class entities has gained increasing attention in the software

architecture research field and also in industrial practice. In recent years, many authors have stressed the importance of thoroughly documenting architecture decisions in software projects (e.g. (Tyree and Akerman 2005, Kruchten 2004a, Tang, Avgeriou, Jansen, Capilla and Ali Babar 2010, Jansen and Bosch 2005)).

Initially, the main perceived benefit of documenting architecture decisions was to share a common understanding of the developed architecture between stakeholders like architects, developers, and customers (Tyree and Akerman 2005, van der Ven, Jansen, Nijhuis and Bosch 2006), primarily to ease change, maintenance, and evolution of the architectural design. Kruchten later stressed that the modeling of (potential) decisions, particularly their dependencies and interrelations, can also support the architect when reasoning about the decisions (Kruchten 2004a). In Chapters 6 and 7, we introduced a documentation framework for architecture decisions that addresses many stakeholder concerns in architecture decisions. Using the conventions of the international architecture description standard ISO/IEC/IEEE 42010 (ISO/IEC/IEEE 2011), the framework provides five viewpoints for architecture decisions, each of which being designed to address different decision-related concerns:

Decision Forces Viewpoint: It makes explicit the relationships between architecture decisions and the forces that influenced the architect when making the decisions out of multiple alternatives. In this context, a *force* is "any aspect of an architectural problem arising in the system or its environment (operational, development, business, organizational, political, economic, legal, regulatory, ecological, social, etc.), to be considered when choosing among the available decision alternatives." (see Chapter 7).

Decision Relationship Viewpoint: It makes explicit the relationships between architecture decisions (e.g. depends on, caused by, or is alternative to).

Decision Chronology Viewpoint: It shows the evolution of architecture decisions over time.

Decision Stakeholder Involvement Viewpoint: It describes the roles of specific stakeholders in the decision-making process, capturing which stakeholders proposed, confirmed, or validated specific decisions.

Decision Detail Viewpoint: It gives detailed information about single architecture decisions, including a comprehensive description of the chosen architectural solution and the rationale for choosing this solution.

Building up on the idea that modeling decisions supports the design process of the architecture, we conjecture that, besides being a useful tool for professional architects, decision viewpoints can guide junior software designers, helping them to make rational decisions. The question is in which areas of software architecting can decision viewpoints help to guide designers.

In this chapter, we report on a comparative multiple-case study conducted with four groups of senior software engineering students (near graduation), to find out if modeling design decisions supports them in following a rational design process. We selected three decision viewpoints from our framework that particularly frame concerns related to decision making support: the *decision detail viewpoint*, the *decision relationship viewpoint* (both from (van Heesch, Avgeriou and Hilliard 2012a)), and the *decision forces viewpoint* (defined in Chapter 7). The results show that particularly the decision forces viewpoint and the decision relationship viewpoint supported the students to systematically identify and evaluate multiple decision alternatives for the design problems.

The rest of this chapter is organized as follows. Section 2 presents the design of the study including the study goal, research conjecture and response variables. Section 3 reports on the data analysis and interpretation. In Section 4, we discuss potential threats to the validity of our findings. We discuss related work in Section 5 and present our conclusions in Section 6.

8.2 Study design

The goal of the study is to explore if designers, who document decision views according to the decision viewpoint framework presented in (van Heesch, Avgeriou and Hilliard 2012a), use more of a rational design approach than designers with an ad-hoc approach. The study is comparative in nature. We try to evaluate the influence of decision view creation on the use of a rational design process.

In cases, where the cause-effect relationship between a specific treatment (in this case the decision view creation) and an outcome (a rational design process) is to be observed, formal experiments can be taken into consideration as empirical method. However, experiments require careful control of the so-called independent variables, which represent potential factors that influence the outcome of the study, to ensure that outcomes are results of the applied treatments. To achieve this control, experiments are usually conducted in a laboratory environment (Wohlin et al. 2012), in which confounding factors can be eliminated, and independent variables can be carefully controlled at pre-determined levels.

Studying the impact of decision view creation (the treatment) on the design process (the outcome) does not allow for this level of control. Apart from the fact that the design process takes multiple weeks, the impact could be wide-ranging, covering multiple aspects of the students' behavior and the project itself. Reducing the measurement to a set of predefined variables would be inappropriate in this case. Additionally, providing a fictional case with artificial requirements and virtual customers would have introduced a threat to validity, as the design of the fictional case could influence the outcome of the study. If conducted as an experiment, the study and its results could be considered as unrealistic and not transferable to industrial practice.

Case studies, on the other hand, are well suited for studying objects of study that are hard to study in isolation (Runeson and Höst 2009). They provide a deeper under-

standing of the situation under study than experiments do. Case studies are suitable for understanding real-life events (Yin 2003). Yin points out that case studies are preferable over experiments in cases, in which control of behavioral events is not possible or not required (Yin 2003). Yet, single case studies are not suitable for doing comparisons, because they are lacking a reference that can be used as a basis for the comparison. This problem has been addressed by Kitchenham et al., who provide guidelines for planning and conducting case studies for the evaluation of software engineering methods (Kitchenham et al. 1995). The guidelines combine the advantages of case study research and formal experiments, and they are well established in the empirical research community (Sjoberg et al. 2007, Easterbrook et al. 2008, Höst and Runeson 2007). In order to allow for the comparison of two software engineering methods, three types of case study arrangements are distinguished:

- Conducting a single case study and comparing the results against a company baseline, for which empirical data is readily available.
- Conducting a single case study using the method for a subset of components, while using a different method for the other components.
- Conducting two case studies, in one of which a new method is applied and compared to the results of the other case study (the so called sister project).

The first type of arrangement is not feasible, because there is no company baseline, against which the students' design activities could be compared. The second arrangement was ruled out, because it would not make sense to ask the students to use the decision framework for some decisions, while forbidding its use for other decisions. In such a scenario, it would have been impossible to avoid maturation effects (Wohlin et al. 2012), e.g. the students would have become more familiar with the problem space or already have a more concrete idea of the overall architecture, when using the other method. Therefore, we decided to apply the third type in our study, thus conducting multiple case studies, in which half of the project teams apply our decision documentation approach, while the other half follows an ad-hoc way of designing and documenting. Recently, several other comparative studies have successfully used case studies to evaluate software engineering methods (Nagappan et al. 2008, Jiang et al. 2008, Serral et al. 2010).

We use general guidelines for conducting and reporting on case studies defined by Runeson and Höst (Runeson and Höst 2009) and synthesize them with the comparison method suggested by Kitchenham et al.; this implies the following additions to the case study method:

- The case study context needs to describe the baseline (gathered from the sister project), against which the impact of the decision viewpoints is compared.
- An evaluation conjecture needs to be defined. Kitchenham et al. use the term *hypothesis*, but we decided to use *conjecture* instead to clearly differentiate from formal experiments.

- Response variables and data collection methods must be defined. The variables correspond to the criteria used to measure the impact of decision viewpoints on the design activities.
- Case variables describing the characteristics of the projects and the development team need to be defined. These variables correspond to independent variables in experimentation, with the difference that they cannot be controlled, but only described in case studies.

Additionally, we used guidelines and suggestions for planning and reporting on case studies (Verner et al. 2009, Brereton et al. 2008).

8.2.1 Context, research goal and conjecture

The study was conducted in the context of the so-called *Software Factories* (SOFA), a lecturing module at the Fontys University of Applied Sciences in Venlo, in the Netherlands. In this module, groups of students work in software projects for external, industrial customers. The students work on their own responsibility; a lecturer, who observes their process, accompanies each of the project teams. After a project runtime of 20 weeks, each of the students is individually assessed by two lecturers. They are graded for their individual performance, the quality of the end product, and the satisfaction of the external customer. None of the researchers was involved in the lecturing module, nor did they have any influence on the grading of the students. The data collected on behalf of the study was not provided to the lecturers, and any publication of the study results takes place after the students received their grades.

The university defines the following project constraints and facilities for the students:

- The students are strongly advised to follow the agile software development process Scrum (Schwaber and Beedle 2002). One of the students takes the role of the product owner. Additionally, they have to write minutes for every team meeting.
- Mandatory use of the project management system Trac (Software 2012), which provides a Wiki, reporting facilities, and a web interface for the version control system Subversion (Tigris.org 2012). Subversion usage is mandatory to store all work artifacts created during the project including source code and configuration files, all project documentation, design artifacts, minutes, and SCRUM specific artifacts like user stories, for instance.
- The students have to work on site at the university for at least three complete working days (8h) per week. Therefore, each of the project teams is provided with its own office, whiteboards, and a beamer.

Using the goal definition technique suggested in the *goal, question, metric* approach (Basili et al. 1994), the overall goal of the study is to:

Analyze the software development processes of senior software engineering students working in groups of 4-6 people **for the purpose of** evaluating the influence of architecture decision view creation **with respect to** reasoning best practices identified in our previous studies (see Chapters 3 and 4) **from the point of view of** external empirical researchers **in the context of** the software factories course at the Fontys University of Applied Sciences in Venlo, the Netherlands.

Based on our previous experience with students who created decision views in their software projects, we derive the following research conjecture from the study goal:

RC: We conjecture that student groups (decision view group) who work in a software project follow a more rational design process if they iteratively create and refine architecture decision views, compared to student groups (comparison group) who follow an ad-hoc approach.

8.2.2 Response variables

In this section, we present the response variables used to determine, which reasoning practices the students follow during the design. We use reasoning best practices, identified in our previous work with students (see Chapter 3) and professional software architects from the industry (see Chapter 4). Each variable poses a question that the study is trying to answer.

Code	**Resp1**
Design activity	Identification of architecture significant requirements (ASRs)
Description	How do the students elicit requirements in general and how do they identify requirements that need to be considered when making architecture decisions?

Code	**Resp2**
Design activity	Requirements negotiation
Description	How do the students negotiate requirements with the project stakeholders? Requirements could be negotiated, for instance, if they unnecessarily impede the project progress, if they are unrealistically challenging, or if they are not state-of-the-art.

Code	**Resp3**
Design activity	Prioritization of requirements
Description	How do the students prioritize requirements when identifying architectural approaches? Attention is drawn in particular to the order and effort put in finding candidate solutions to address specific requirements.

Code	**Resp4**
Design activity	Documentation of requirements
Description	How do the students document requirements? Particular attention is payed to the S.M.A.R.T. characteristics (Mannion and Keepence 1995) *specific*, *measurable*, *attainable*, *realizable*, and *traceable*

Code	**Resp5**
Design activity	Discovery of design options
Description	How do the students identify design options to address architectural problems?

Code	**Resp6**
Design activity	Balancing advantages and disadvantages of design options
Description	How do the students consider the advantages and the disadvantages when selecting a solution out of multiple design options during architectural evaluation (with architectural evaluation, we refer to the process of choosing out of multiple design options, as defined in (Hofmeister et al. 2007))?

Code	**Resp7**
Design activity	Discussion of multiple design options in combination
Description	Do the students discuss multiple architectural approaches in combination?

Code	**Resp8**
Design activity	Avoidance of unnecessary complexity
Description	Do the students proactively take measures to avoid unnecessary complexity in the architectural design?

Code	**Resp9**
Design activity	Validation of design options against the ASRs
Description	How do the students validate design options against the architecture significant requirements during architectural evaluation? This variable includes the making of compromises in cases where a design option has conflicting influences on multiple ASRs.

Code	**Resp10**
Design activity	Prototyping of design options
Description	Do the students build prototypes, and if so, what are they used for?

Code	Resp11
Design activity	Evaluation of the architecture as a whole
Description	How do the students evaluate their designed architectures?

8.2.3 Case variables

In the following, we define case variables concerning the software projects and the participants of the study:

Code	CaseVar1
Name	Study group
Description	This variable describes, if the students in a project document decision views during the design (decision view group), or not (comparison group).
Scale Type	Nominal
Unit	n.a.
Range	'decision view group','comparison group'

Code	CaseVar2
Name	Programming experience
Description	The programming experience is one of the measures used to describe the software engineering experience of the subjects. The effect of this variable on the outcome of the study is reduced by the fact that the students in the two study groups are balanced regarding their programming experience. We take both, industrial programming experience and academic experience into account.
Scale Type	Ordinal
Unit	Years
Range	4 classes: 0 years, 1-3 years, 3-7 years, >8 years

Code	CaseVar3
Name	Design experience
Description	Similarly to the programming experience, the design experience of the students could have an effect on the outcome of the study. The effect of this variable on the outcome of the study is reduced by the fact that the students in the two study groups are balanced regarding their design experience.
Scale Type	Ordinal
Unit	Years
Range	4 classes: 0 years, 1-3 years, 3-7 years, >8 years

Code	**CaseVar4**
Name	Industrial experience
Description	The industrial experience is expressed as the number of years, the students have worked as a software engineer in the industry (i.e. not in an academic context); for instance in a side job, or prior to the study. The effect of this variable on the outcome of the study is reduced by the fact that the students in the two study groups are balanced regarding their industrial experience.
Scale Type	Ordinal
Unit	Years
Range	4 classes: 0 years, 1-3 years, 3-7 years, >8 years

Code	**CaseVar5**
Name	Project domain
Description	The domain of the projects could have an influence on the design activities, as some domains like healthcare or embedded systems require designers to think more carefully about safety critical decisions.
Scale Type	Nominal
Unit	n.a.
Range	Possible values: Avionics, Command and Control, Embedded Systems, E-Commerce, Enterprise Computing, Finance, Healthcare, Realtime, Manufacturing, Software Engineering, Scientific, Simulation, Telecommunication, Transportation, Utilities, Marketing, Logistics, Web Applications, Others

Code	CaseVar6
Name	Difficulty of the project
Description	The difficulty of the SOFA projects could theoretically influence the design activities followed by the students. Difficulty, in this context, refers to the difficulty of the problem. Judging the difficulty of a project based on objective metrics is challenging and vulnerable. Therefore, we decided to estimate the difficulty of the projects by asking the four lecturers, who supervise and grade the SOFA projects, to rate the difficulty of each SOFA project. The lecturers were asked to take into consideration the project goals, technologies that would have to be used, as well as the students' previous knowledge in the project domains. Each of the lecturers was knowledgable about two projects, because they acted as a supervisor for one project, and as assessor for another project. In addition, the researchers judged the difficulty of the projects, using the same criteria as the lecturers. The difficulty of the projects was then calculated by taking the median value of the three given ratings (supervising lecturer, assessing lecturer, and researchers)
Scale Type	Ordinal
Unit	n.a.
Range	5-point Likert scale: 1 for *very simple* to 5 for *very difficult*

Code	CaseVar7
Name	Experience in the project domain
Description	This variable refers to the experience of the students in the domain of the respective SOFA project (variable CaseVar5). The domain experience could for instance have an influence on the effort in or intensity of exploring the problem and solution spaces.
Scale Type	Ordinal
Unit	Years
Range	4 classes: 0 years, 1-3 years, 3-7 years, >8 years

8.2.4 Cases, objects and subjects description

In this section, we explain the four cases. Additionally, important characteristics of the subjects, the sampling procedure, and the object under study will be further elaborated.

Cases and objects

In total, we observed four different software projects run as part of the Software Factory module. In the following, each project will be briefly described. The customer of one of

the projects asked for anonymity. As a consequence, we decided to use pseudonyms for all projects.

Project alpha: This project is a brown-field, dealing with a legacy text system, which is used to dynamically generate multiple types of documents based on templates and information stored in a database. Using the templates, data can be composed in multiple ways before being assembled into document formats like PDF, for instance. The Bavarian Department of Justice is one of the prominent users of the system. The primary task of this project team is an architectural re-design to a service oriented architecture (SOA). The customer of the project, a medium-sized german software company, wants to migrate all business-critical services to SOA in the long term.

Project beta: The customer of this project is a dutch personnel service for chefs (cooks)[1], specialized on temporary arrangements like catering, cook workshops, or interim executive chefs. The primary task of the SOFA project team is the development of a software platform for online personnel services in the gastronomy business, where freelancing cooks can register and apply for jobs, which are posted to the site by restaurant owners, for instance. The project is a green-field; all technology choices must be made by the SOFA participants. The customer himself does not have a software engineering background.

Project gamma: In project gamma, the students were given the task to extend an existing standalone client application for geo-marketing in the areas of sales, marketing and controlling. The extension must be capable of displaying different location based information in a geographical map. Data must be retrieved from a central XML repository, which can be queried using a proprietary object-oriented query language. The customer of the project is a geo-marketing consultancy in Germany, which, among others, maintains its own geo-marketing software tools.

Project delta: Project delta is a green-field project. The customer is a traditional family-operated rose-growing company in the Netherlands, who operates mainly on the international container market. The goal of the SOFA project team is the development of an addition to the customer's enterprise resource planning (ERP) system, which is capable of processing information gathered from RFID tags, which will be attached to the different types of rose transportation devices, repositories and gates. Apart from scanning RFID tags, data needs to be gathered using a web application and synchronized with an existing data repository.

Note that two of the projects require software systems to be developed from scratch (projects beta and delta), while the remaining two projects are evolutionary in nature. These characteristics were considered when assigning the projects to either the decision view group, or the comparison group, i.e. each of the study groups has one green-field

[1] A company that acts as a broker between restaurant owners and the searched cooking personal.

and one brown-field project. Please refer to Section 8.2.4 for the details of the study group assignment.

Subjects and sampling

Using students in empirical studies is a sensitive issue that obliges researchers to take a number of ethical and epistemological factors into account. On the one hand, studies with students are often criticized for not being generalizable (Svahnberg et al. 2008); on the other hand, researchers should make sure that the study has as much pedagogical value for the participating students as possible. To make sure that these factors were sufficiently taken into account, Carver et al.'s checklist for conducting empirical studies with students (Carver et al. 2010) was used as a guideline for the design of this study. In the following, we list all items of this checklist together with a brief explanation on how the checklist item was considered:

1. **Ensure adequate integration of the study into the course topics** – The research goal was to study the effect of decision documentation on the design process of junior software designers. The educational goal of the study was two-fold: The students should become aware of problems in their decision making processes, and be provided with concrete ways to tackle these problems. The main educational goal of the SOFA project is to familiarize students with realistic software projects, in which they have to make all design decisions themselves (i.e. without assistance by lecturers), communicate with the customer, and take over responsibility for their end-products. Therefore, by conducting the study in a course, in which the students have to work in project teams to solve a real-world case, the study was properly integrated into the course topic.

2. **Integrate the study timeline with the course schedule** – The timeline for the study was explicitly planned according to the start of the SOFA project. The first seven weeks of the project were observed, because naturally, the most design decisions had to be made in the first half of the SOFA semester, while the second part would be primarily spent on programming and report writing.

3. **Reuse artifacts and tools where appropriate** – The tools and artifacts gathered in the study were all part of the SOFA course. Apart from decision views, the students did not have to use additional tools or create additional artifacts for the purpose of the study.

4. **Write up a protocol and have it reviewed** – A study protocol was written before the study and reviewed among the authors in multiple iterations. In addition, the study was discussed with the five lecturers of the course to make sure that it aligns with the course and makes no unrealistic assumptions.

5. **Obtain subjects' permission for their participation in the study** – At the beginning of the SOFA course, the students were informed about the plan to conduct

an empirical study in the context of the module. In particular we explained which data we would collect and assured them that no information would be shared with their course lecturers. The students were not informed about the concrete goal of the study, because this could have biased the results. They were given the opportunity to withdraw from the study without giving further reasons, e.g. by sending an e-mail to one of the researchers.

6. **Set subject expectations** – Prior to the study, the students were informed about the purpose of the study, the time they would need to invest and the benefits they can expect from the study, i.e. information about and suggestions for improving their design processes.

7. **Document information about the experimental context in detail** – This research report contains detailed information about the experience of the subjects, the nature of the SOFA course, and the concrete projects run as part of the SOFA.

8. **Implement policies for controlling/monitoring the experimental variables** – The study variables, as well as the data collection methods and data sources used to monitor the variables are described in detail in this study report.

9. **Plan follow-up activities** – At the end of the semester, one of the researchers presented the preliminary results of the study to the students. On this occasion, the students were informed about the concrete goals of the study and the research conjectures, namely that the project teams who documented decision views would be expected to follow a more rational decision making process. The study design was also discussed with the students as well as potential threats to validity. That way, the students also learned something about conducting case studies, which was especially interesting, because the students had been following a course on applied research methods as part of their curriculum.

10. **Build or update a lab package** The collected data was assembled in a study database (as proposed in (Yin 2003)), which was used as a basis for the analysis and prepared for reuse in future studies.

The participants of the study were selected using convenience sampling (Given 2008); all students who took the SOFA course in the winter semester 2011/2012 were invited to participate. None of them refused. In total, 21 students took part in the study. The researchers were not given the opportunity to intervene in the assignment of students to one of the four SOFA projects introduced before. Each student chose a project based on personal interests.

Assignment of projects to study groups

As mentioned before, we were not given the opportunity to assign students to the four available projects, hence we could only assign the four project teams to the study groups

Table 8.1: *Descriptive statistics used for assigning projects to study groups*

	Decision View Group		Comparison Group	
	Project Alpha	Project Beta	Project Gamma	Project Delta
Avg. Programming experience (months)	49,2	56,8	68,4	82
Avg. Design exp (months)	32,2	50,33	43	74
Avg. Industrial exp. (months)	7	25,18	31,83	19,28
Primary domain(s)	Web Applications	Web Applications	Marketing	Web Applications, Logistics
Avg. exp. primary domain(s) (months)	13,4	31,2	13,33	32,5
Median difficulty of project	3	4	3	3
Greenfield (GF) or brownfield (BF)	BF	GF	BF	GF

(i.e. decision view group or comparison group) in a way that both study groups were balanced with respect to the relevant case variables, as far as possible. The following case variables were taken into consideration. First of all, the two study groups should be balanced with respect to green-field and brown-field projects (see Section 8.2.4). Second, the difficulty of the project tasks should be comparable in both study groups (CaseVar6, Section 8.2.3). Finally, the students' industrial experience, as well as previous experience regarding programming, architecture, and the domain of the project should be balanced in both study groups as far as possible (CaseVar2,3,4, and 7, Section 8.2.3).

Table 8.1 shows descriptive statistics for the variables used as a basis for the assignment of SOFA project teams to study groups[2]. The data was gathered using a web questionnaire. For the estimation of the difficulty of the four projects, we used the procedure described for CaseVar6 in Section 8.2.3. See Table D.1 for the detailed ratings given by the four lecturers and the researchers. Table 8.1 shows only domains that were selected by the majority of students in each project (referred to as primary domain). In the case of project delta, two domains were equally often selected. The average domain experience refers only to the primary domain; in case of project delta, the average was calculated from both primary domains.

The most important project characteristic for the study group division was the current state of the software project; i.e. a new project starting from scratch (green-field project), or an existing project that is further developed (brown-field project). Thus, we had to assign one brown-field project to each of the study groups. Both brown-field projects were similar with respect to domain experience and difficulty, but the members of project gamma were more experienced regarding programming, design, and industrial experience; therefore, we decided to assign project gamma to the comparison group. This was mainly to exclude the previous experience as a confounding factor with respect to the rationality of the decision making processes; if we had assigned the more experienced project to the decision view group, the more rational decision making process could have resulted from the previous experience, rather than from the documentation of decision views.

Between the two green-field projects, we decided to assign project beta to the deci-

[2]More detailed statistics about the variables can be found in Appendix D.2

sion view group and project delta to the comparison group. In this case, the advance of programming and design experience on the side of project delta compensates the difference of approximately six months with respect to industrial experience.

The response variables were measured mainly in the first seven working weeks. A final round of focus groups with all four project teams was conducted at the end of the SOFA semester in January 2012. Please refer to Section 8.2.5 for the details of the data collection.

The decision view group received a two-hour training on creating the decision views additionally to written guidelines and examples. They were also provided with an MS Word template for the detail view, an MS Excel template for the decision forces view, and a Visual Paradigm template containing an example of a relationship view. All members of the decision view project teams were obliged not to talk to the members of the other two project teams about decision views. The comparison group was not informed about decision views at all.

8.2.5 Instrumentation and data collection procedures

In this section, we describe the data collection methods used, the data sources, and their mapping to the study variables. In qualitative research, it is important to develop ideas not only based on one data source using one specific data collection method. Triangulation of data sources and data collection methods has been a good practice for qualitative researchers to make sure that there are multiple forms of evidence to back up a conclusion rather than single data points or very few incidents (Creswell and Miller 2000, Lethbridge et al. 2005, Yin 2003). Patton differentiates four types of triangulations (Patton 2002): a) data source triangulation, b) investigator triangulation, c) theory triangulation, and d) methodological triangulation. In this study, we apply data source triangulation (collecting data from more than one source) and methodological triangulation (collecting data using different methods). Our motivation is to have at least two data sources, and their corresponding data collection methods, for all conclusions drawn from the collected evidence.

We use the data collection classification scheme from (Lethbridge et al. 2005) to describe the methods used in this case study:

- **Questionnaire**: At the beginning of the study, the students filled in a questionnaire to gather information about the nature of the SOFA project they chose, their previous experience in software engineering activities, and their experience in the domain of the chosen SOFA project.
- **Work diaries**: As part of the SOFA course, the students had to create daily work diaries, in which they document their process, the decisions they made, and the tasks they identified. These diaries (also referred to as team minutes in the remainder of this chapter) were made available to the researchers.
- **Documentation analysis** (study of work artifacts): The Subversion and Trac servers, which all project teams had to use, were accessible for the researchers.

Table 8.2: *Mapping of variables to data collection methods*

	Questionnaire	Work diaries	Documentation analysis	Focus Groups
Resp1: Identification of ASRs		X	X	X
Resp2: Requirements negotiation		X	X	X
Resp3: Prioritization of requirements			X	X
Resp4: Documentation of requirements			X	X
Resp5: Discovery of design options		X	X	X
Resp6: Balancing advant. and disadvant.			X	X
Resp7: Discuss mult. options in comb.		X	X	X
Resp8: Avoid complexity		X	X	X
Resp9: Validate options against ASRs		X	X	X
Resp10: Prototyping options		X	X	X
Resp11: Evaluation of arch. as a whole		X	X	X
CaseVars2-4: Previous experience	X			
CaseVar 5: Project domains	X		X	
CaseVar6: Difficulty of the project		X	X	
CaseVar7: Experience in domain	X			

All working artifacts that were checked in by the different project teams (e.g. requirements and design documents, minutes, source code, and the decision views) were collected and analyzed. In addition, one person of every project team was responsible for taking pictures of all whiteboard sketches the project teams created.

- **Focus Groups**: Weekly focus groups (30-60 mins) with each of the project teams about the design process and the decisions made (audio recorded and transcribed). In the focus groups, the participants were encouraged to talk freely about their design, the progress they made, and the process they have been following. Additionally, in the decision view project teams, the documented decision views were also reviewed. In the comparison group projects, the participants were interviewed without explicitly mentioning the notion of architecture decisions. In addition to the weekly focus groups, a final round of focus groups was conducted with each of the project teams at the end of the SOFA semester. Furthermore, the focus group moderator took field notes, in which he noted down impressions gained during the focus groups. Field notes are important to complement the audio recordings, as some important information is non-auditory, e.g. supporting a teammate's comment by nodding, or strong opposition expressed only in body language. Field notes can also be used to write down initial ideas about the project process that can serve as focus points during the data analysis, e.g. "It seems that that project alpha did not align the user stories with the functional requirements gathered before".

The field notes and all collected data were stored in a digital study database.

Table 8.2 shows a mapping of study variables to data collection methods. Note that

there are at least two data sources for each response variable. The data for the case variables was mainly collected using the questionnaire at the beginning of the study.

8.2.6 Analysis procedure

The gathered data was analyzed qualitatively using grounded theory (Glaser and Strauss 1967). In grounded theory, theories are developed by systematically analyzing the collected data, and constantly comparing findings to the previous conjectures. It is thus a very labor-intensive, iterative process, in which theories evolve slowly and are always grounded in the collected data. In recent years, the use of grounded theory in software engineering has become acceptable (Urquhart et al. 2010, Adolph et al. 2011); conceivably, because contemporary research in the field seeks to involve more exploratory research, rather than relying on purely confirmatory studies.

One might suggest that grounded theory is not appropriate in cases where researchers seek to verify a research hypothesis (Urquhart et al. 2010), i.e. the research is confirmative rather than exploratory. In this particular case, however, despite of the formulated research conjecture, the research is fundamentally exploratory: we attempt to gain a broad understanding of how the students design software and how the documentation of different decision views during the design process influences the design activities. This explorative part of the study is a prerequisite for the comparison of the two study groups.

Qualitative analysis approach used

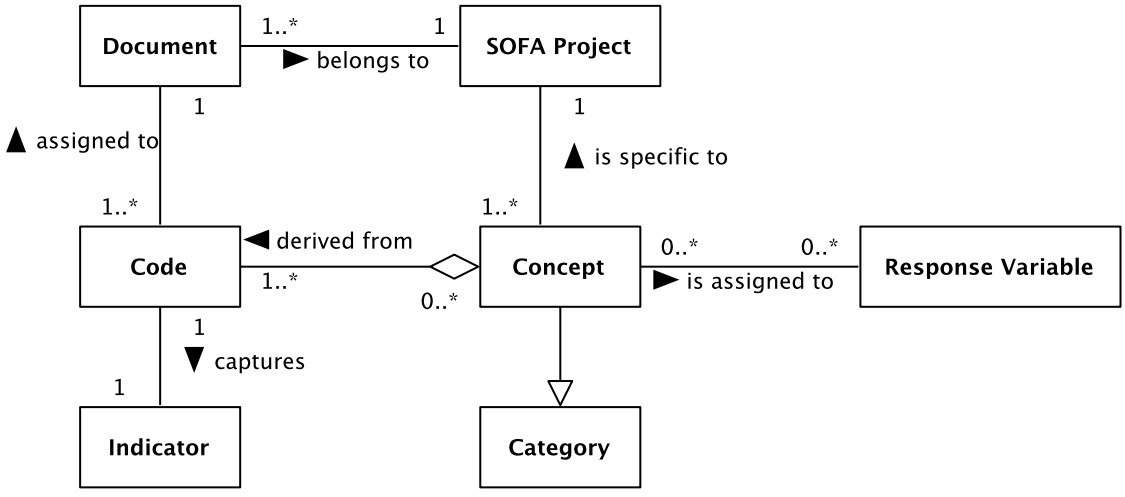


Figure 8.1: *Conceptual model of grounded theory entities*

In the following, we describe the qualitative analysis procedure as applied in this study. The detailed description allows other researchers to assess the quality of the analysis process or to adopt it in own studies. Figure 8.1 shows a UML class diagram summarizing the conceptual entities of the qualitative analysis and their relationships respectively; they will be discussed in the following steps.

Step1 - Filter study documents: In the first step, we browsed all documents that had been collected in the study database during the course of the study. Besides transcripts, field notes, and minutes, this process included all documents that were created by the SOFA project teams and uploaded to their respective subversion repository or Trac wiki.

If the content did not have any relation to the decision-making process, the document was excluded; all other documents were taken to the next steps, described below. In total, 401 documents were browsed in step one, 254 of which were found to be relevant. The excluded documents contained documentation of used third party software, or bash scripts and latex templates, for instance.

Step 2 - Normalization: In the next step, the chosen documents from step one were normalized: each file was converted to PDF, and renamed to express the name of the SOFA project, the type of file, the original file extension, and the date at which it was downloaded. The result of this step was a number of PDFs assigned clearly to one of the four SOFA projects.

Step3 - Coding: In step three, the documents from each SOFA project were coded. During this procedure, the documents were carefully studied and each phrase, sentence, or paragraph that indicated a certain behavior (called *indicator* in grounded theory literature (Strauss 1987)) was labelled with a code. This approach to coding is originally referred to as *open coding* (Corbin and Strauss 2008), used to generate the concepts that become the basis for further analysis (see step 4).

As opposed to other researchers, who suggest to assign one-word codes to express indicators (e.g. (Adolph et al. 2011)), we chose to use brief statements as codes. We made this decision, because we experienced that finding single words to clearly express an indicator is extremely challenging and forces the analyst to read large passages over and over again, when comparing codes to previously assigned codes. Brief statements allow to be much more expressive. We used PDF annotations to assign the codes. That way the codes are shown next to the text without disturbing the flow of reading. PDF annotations have the additional advantage that codes can easily be revised and different analysts can assign different codes.

To collect the codes from the documents and to support the constant comparison process, we developed a software that registers documents, collects all codes and stores them in a study database using the model shown in Figure 8.1 as domain model. The constant comparative method in grounded theory obliges the researcher to frequently step back to analyze all collected codes and to compare new codes to existing ones to develop a theory. The tool we developed supports this process, as previously assigned codes from other documents are permanently shown to the analyst while coding additional documents. This enables the analyst to identify commonalities in codes, which is useful for discovering concepts.

Step4 - Identify concepts: Steps three and four were repeated in multiple iterations when analyzing the documents of one SOFA project. In step four, the previously gathered codes were compared to identify common concepts. Here, a concept is a representation of a pattern of behavior, suggested by a set of indicators, which on their part are captured using codes (Adolph et al. 2011). The concepts were assigned using the previously mentioned software, we developed. During the analysis procedure, the concepts slowly evolved; they had to be revised regularly after additional documents had been analyzed.

Step5 - Assign concepts to variables: After finishing the coding and the declaration of concepts, each concept was assigned to one or more response variables.

Step6 - Classify concepts into general categories: After finishing steps one to five, we had defined a set of concepts describing the behavior of each project team with respect to the response variables. The concepts are specific to projects, i.e. they summarize multiple codes from the documents of one project team. In order to compare the results from the different project teams, we analyzed the concepts and classified them into categories. A category, in our understanding, is a project independent abstraction of one or more concepts from potentially different projects. This is in line with Glaser, who describes a category as a concept used on a higher level of abstraction (Glaser 1998). Figure 8.2 illustrates the relationship between categories and project-specific concepts. The categories, defined in this study, can be found in Table 8.3.

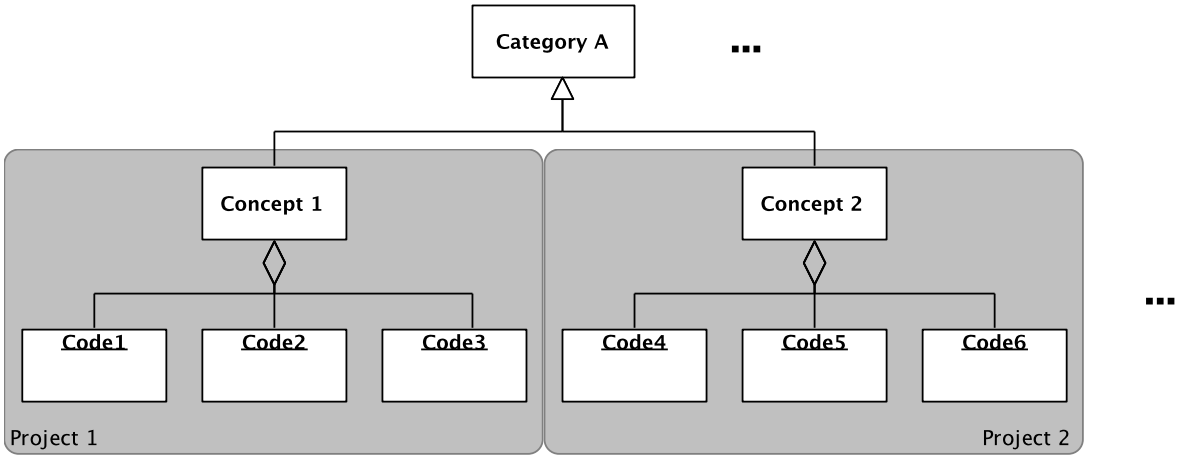


Figure 8.2: *Comparing projects using categories of concepts*

8.3 Analysis and interpretation

In this section, we present the results of the qualitative analysis with respect to commonalities and differences between the projects in the two study groups. The section

is organized according to the response variables. Please refer to Appendix D.3 for a detailed list of concepts and categories derived from the collected data. The codes and concepts, which are the result of steps three and four in the analysis, are not listed here for reasons of space; in total, more than 620 codes were assigned to the various documents resulting in 165 concepts.

Table 8.3: Categories

Code	Category
Cat1	Systematically searched for multiple design options.
Cat2	Conducted research to identify design options.
Cat3	Most design options are technology related.
Cat4	Followed a reuse over reimplementation strategy.
Cat5	Research tasks regarding design options for a decision point were divided among the group.
Cat6	Developed overall vision of the architecture to identify decision points.
Cat7	Always chose first viable solution.
Cat8	Most gathered requirements are functional.
Cat9	Non-functional requirements were not actively elicited.
Cat10	Actively explored the functional problem space.
Cat11	No explicit distinction between architecturally relevant requirements and other requirements.
Cat12	Actively involved to understand the business domain.
Cat13	The group tried to install and run the existing software as a first step in the analysis.
Cat14	Multiple types of documentation used for requirements.
Cat15	Systematically clarified vague requirements with customer.
Cat16	Responsibility for describing requirements is silently transferred to the customer.
Cat17	Requirements slowly emerged during the design phase.
Cat18	No clear separation between requirements and resulting design or implementation tasks.
Cat19	Quality attribute requirements were not documented.
Cat20	Group does not gain a collective understanding of the requirements.
Cat21	Scrutinized requirements with respect to feasibiliby and usefulness.
Cat22	Requirements were not called into question.
Cat23	Proposed additional requirements.
Cat24	Negotiated blocking requirements.
Cat25	Challenging requirements were prioritized.
Cat26	Requirements are addressed in no recognizable order.
Cat27	Requirements were not described well. Single words or brief statements used without explanation.

Table 8.3 – continued from previous page

Code	Category
Cat28	Explicitly discussed pros and cons of all major design decisions.
Cat29	Conducted research to find arguments in favor of and against design options.
Cat30	Group members challenge each others arguments a lot.
Cat31	Decisions are mostly made without explicit reasoning.
Cat32	Most decisions are not discussed in the group.
Cat33	Technological dependencies were systematically explored before making decisions.
Cat34	Many technological decisions were made in combination.
Cat35	Avoiding complexity was an explicit goal of the group.
Cat36	Validated technology options against ASRS.
Cat37	No indicators for an explicit consideration of ASRs when making decisions.
Cat38	We aware that trade-offs could be necessary.
Cat39	Made trade-offs between multiple requirements (very rarely).
Cat40	Used prototypes to understand technological options.
Cat41	Prototypes mainly used to learn how the technology can be used.
Cat42	Prototypes were used to estimate the influence of a design option on quality attribute requirements.
Cat43	Permanently maintained an overview over the complete system.
Cat44	Architecture was not evaluated as a whole.

Table 8.4 shows the categories as assigned to the four projects and response variables respectively. It was taken as a basis for the subsequent analysis. Cases, in which a category was assigned to each of the four projects, are regarded as a commonality; cases, in which a category was assigned to the two projects within one study group, but not in the two projects from the other study group, are regarded as a difference. In the latter case, we discuss in how far the difference results from the decision view creation. Cases, in which a category was assigned to a single project only, or cases in which a category was assigned to one project in the decision view group and one project in the comparison group are not discussed, because they do not allow drawing conclusions with respect to the impact of using the decision viewpoints on the design activities.

Table 8.4: Categories assigned to projects and response variables

		Decision View Grp.		Comparison Grp.	
Resp. var.	**Categ.**	**Alpha**	**Beta**	**Gamma**	**Delta**
Resp1: Identification of ASRs	Cat8		X	X	X
	Cat9	X	X	X	X
	Cat10		X		
	Cat11	X	X	X	X
	Cat12		X		X
	Cat13	X			
	Cat15		X		
	Cat16			X	X
	Cat17	X			X
	Cat18	X		X	
	Cat19	X		X	X
	Cat20				X
Resp2: Requirements negotiation	Cat21		X		
	Cat22			X	X
	Cat23		X		
	Cat24	X	X		
Resp3: Prioritization of requirements	Cat25	X	X		
	Cat26			X	X
Resp4: Documentation of requirements	Cat14	X	X	X	X
	Cat27	X	X	X	X
Resp5: Discovery of design options	Cat1	X	X		
	Cat2	X	X		
	Cat3	X	X		
	Cat4		X		
	Cat5	X	X		
	Cat6	X			
	Cat7			X	X
Resp6: Balancing pros and cons of design options	Cat28	X	X		
	Cat29	X	X		
	Cat30	X	X		
	Cat31			X	X
	Cat32			X	X
Resp7: Discussion of multiple design options in combination	Cat33	X	X		
	Cat34	X	X		
Resp8: Avoidance of complexity	Cat35	X	X		
Resp9: Validation of design options against the ASRs	Cat36	X	X		
	Cat37			X	X

Table 8.4 – continued from previous page

		Decision View Grp.		Comparison Grp.	
Resp. var.	**Categ.**	**Alpha**	**Beta**	**Gamma**	**Delta**
	Cat38	X	X		
	Cat39	X			
Resp10: Prototyping design options	Cat40	X	X	X	X
	Cat41	X		X	X
	Cat42		X		
Resp11: Evaluation of architecture as a whole	Cat43	X	X		
	Cat44	X	X	X	X

8.3.1 Resp1 - Identification of ASRs

None of the four project teams actively elicited non-functional requirements (Cat9). In some cases, the students even ignored hints given by the customer with respect to quality attribute requirements. All project teams focused on functional requirements, which were mainly understood as use cases, or user stories.

Generally, the four project teams also did not make an explicit distinction between requirements in general, and architecturally relevant requirements (Cat11). The only exception were the forces views, which triggered the two project teams in the decision view group to select only those requirements that were architecturally relevant. In all other documents containing requirements, this distinction was not evident.

While the two project teams in the decision view group actively approached the customer multiple times to elicit and clarify requirements, the two project teams in the comparison group transferred the responsibility for defining requirements completely to the customer (Cat16). They took no effort to elicit requirements additionally to an initial list of requirements they received from the customer. In case of project delta, the students neglected requirements elicitation, although the customer explicitly told them in the beginning that he expected them to do a thorough analysis of the project domain and the resulting requirements. Both project teams in the comparison group also took no effort to clarify requirements they had not understood well; they rather speculated about their meaning internally during the project meetings. The difference regarding the active elicitation of requirements was potentially caused by the decision forces view, which requires students to actively reflect on requirements and other forces that influence their design decisions.

8.3.2 Resp2 - Requirements negotiation

There is a notable difference on how the two study groups negotiate requirements. The two project teams in the comparison group did not question any requirements (Cat22), the project teams in the decision view group actively went into requirement discussions

with the customers. Both project teams in the decision view negotiated requirements they experienced as unnecessarily constraining or even blocking (Cat24). To give an

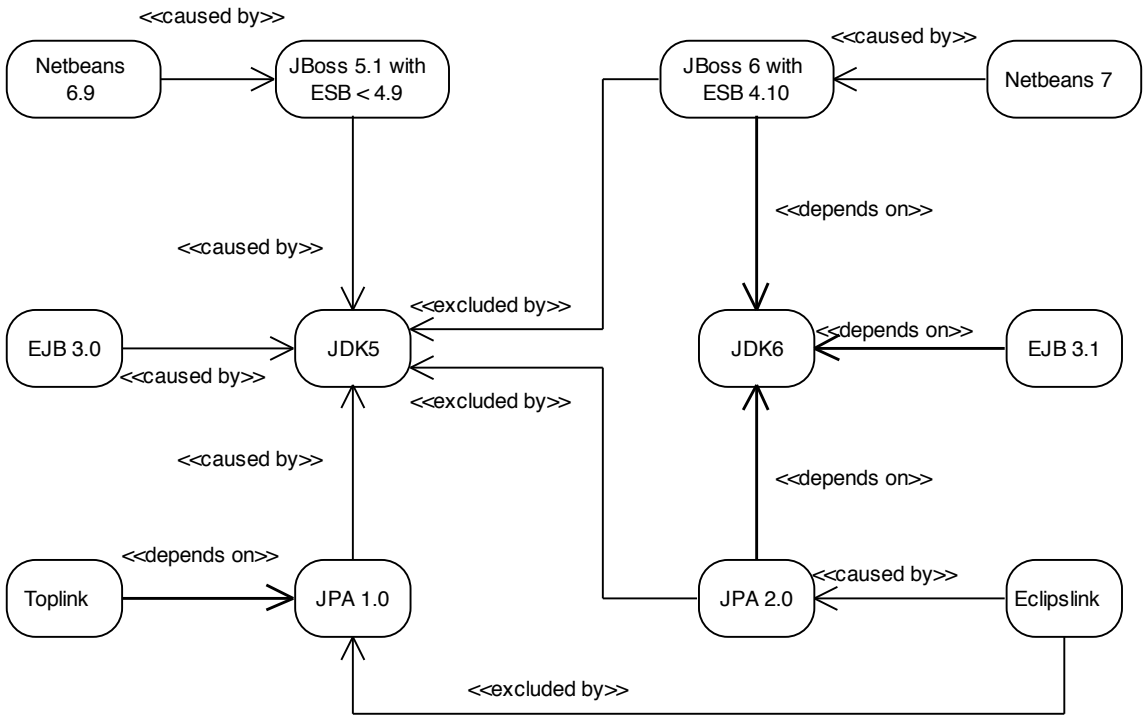


Figure 8.3: *Relationship view created by project team alpha to understand the impact of a technological constraint*

example, the customer obliged project team alpha to use JDK 1.5, a rather old Java Development Kit, because one of the third party libraries used by the customer was not compatible with newer Java versions. This technological constraint turned out to have a huge impact on the design options that could be taken into consideration; particularly on the choice of the enterprise service bus (ESB) technology. Most recent ESB implementations require a JDK greater than 1.5, which would have forced the students to rely on older implementations and at the same time older versions of the Java enterprise edition (JEE). This, however, would have prevented the usage of frameworks, which require newer versions of JEE. To understand the true dimensions of the JDK 1.5 constraint, the students created a decision relationship view showing the technological choices they would have made without the constraint, and the technological choices they could make regarding the constraint (see Figure 8.3). Using this relationship view, the customer could be convinced to drop the constraint.

Apart from this example, the decision forces view seemed to create a much more critical attitude in the decision view group towards the requirements compared to the comparison group, because it forced the two project teams in the decision view group to actively reflect on requirements and the design options that can possibly satisfy them. The two project teams in the comparison group took the requirements for granted (i.e. they did not question them). To make matters worse, they also did not make sure that the decisions they made were consistent with the requirements, as we will explain later.

8.3.3 Resp3 - Prioritization of requirements

Another difference between the decision view group and the comparison group was observed with respect to the prioritization of requirements. The project teams in the decision view group prioritized requirements (Cat25). Requirements, recognized as being challenging or very important, were given a higher priority than other requirements. As an example, project team alpha immediately started searching for technological design options to realize a service oriented architecture, while other requirements regarding the realization of a full-text search and a tag cloud were given such a low priority that they could not be implemented until the end of the project. Project team beta put priority on finding a web framework that would allow to add and change content easily. Requirements regarding social media and third party payment provider integration, in accordance with the customer, were given a lower priority.

As opposed to the decision view group, the project teams in the comparison group did not address requirements in a recognizable order (Cat26). Although in both projects, some of the requirements were clearly more challenging than others, these requirements were not given a higher priority. As an example, in project delta, the students had to make sure that RFID scanners would work partially in an unfriendly environment (outside, exposed to dirt, whether, heat, and cold temperatures), using different types of available networks (e.g. LAN, WIFI, and GPRS) without losing data. Until the end of the project, the students ignored these requirements. In project gamma, requirements were chosen based on the personal interests of project members, instead of estimating their importance systematically.

8.3.4 Resp4 - Documentation of requirements

In all four projects, we observed that different types of documentation were used to capture requirements (Cat14). In addition, none of the project teams assembled requirements in a central place. The project teams alpha and delta captured initial requirement statements, made by the customer, in a word document; all four teams documented some functional requirements using use cases stored at different places in the projects' repositories; project teams beta and gamma additionally documented user stories (a SCRUM-specific way of documenting customer requirements) in the Trac systems. In the decision view projects, additional requirements and forces were captured in the forces views. When comparing the requirement statements made in the different documents, inconsistencies were found in all four projects. For instance, some use cases documented by projects beta and gamma were not documented as user stories, whereas other functional requirements only existed in the form of user stories, but not in the form of use cases.

Another phenomenon observed in all four projects is the fact that the different types of requirements documents were not revised or updated any more. This is a strong indicator for the lack of a thorough requirements management process. In the focus groups, all four project teams acknowledged that requirements kept changing, but they also

admitted that existing requirements documentation was not systematically adapted to those changes. Therefore, the existing requirements documentation was quickly outdated. As a consequence, the students in all four projects were not able to gain a holistic understanding of all relevant requirements; nor could they systematically regard all requirements in the design process.

Another major issue commonly identified in all projects was the fact that the documented requirements did not have the SMART (Mannion and Keepence 1995) characteristics. The students mainly used brief statements, sometimes only single words, to express requirements (Cat27). An example from project alpha, which was not further elaborated, is "Fulltext search should be possible". An example from project delta is "assigning products to Ferro / Pallet Tags", or simply "Store data". Our initial intention, to analyze the documented requirements with respect to how SMART they are was dropped, because the quality of the requirements documentation was so low that a further analysis for the purpose of comparison would not have made sense.

The findings regarding Resp4 suggest that the decision viewpoints did not help the students to systematically document and manage requirements.

8.3.5 Resp5 - Discovery of design options

As opposed to the comparison group, the decision view project teams systematically searched for multiple design options (Cat1). To identify design options, the two project teams conducted research (Cat2) using the Internet. Already at an early stage, the decision view project teams developed a vision of the overall architecture to identify decision points (see Figures D.5 and D.6 for two examples) and then assigned different team members to conduct research regarding design options for a specific decision point (Cat5). Both project teams considered at least two design options for each major decision point. It was evident that most design options were technology-related (Cat3). Only in rare cases, the project teams considered the use of a design or architectural pattern, for instance.

In contrast to the decision view group, the two project teams in the comparison group did not systematically search for design options before making decisions. Even when being asked directly about considered alternatives, they acknowledged that they only searched until they found a viable solution (Cat7) and then moved on to another decision. In case of project delta, the students used mainly trial and error. Only when a chosen design option turned out to be the wrong choice during the implementation, they hastily searched for an alternative.

8.3.6 Resp6 - Balancing advantages and disadvantages of design options

A significant difference between the two study groups is how design options were analyzed and compared. The project teams in the decision view group heavily used the forces view to explicitly discuss pros and cons of major design decisions (Cat28). The

project teams created partial forces views to support single decisions, and kept revising one central forces view with all major design decisions. Figure 8.4 shows a forces view created by project team beta to support the choices of a programming language and a database management system.

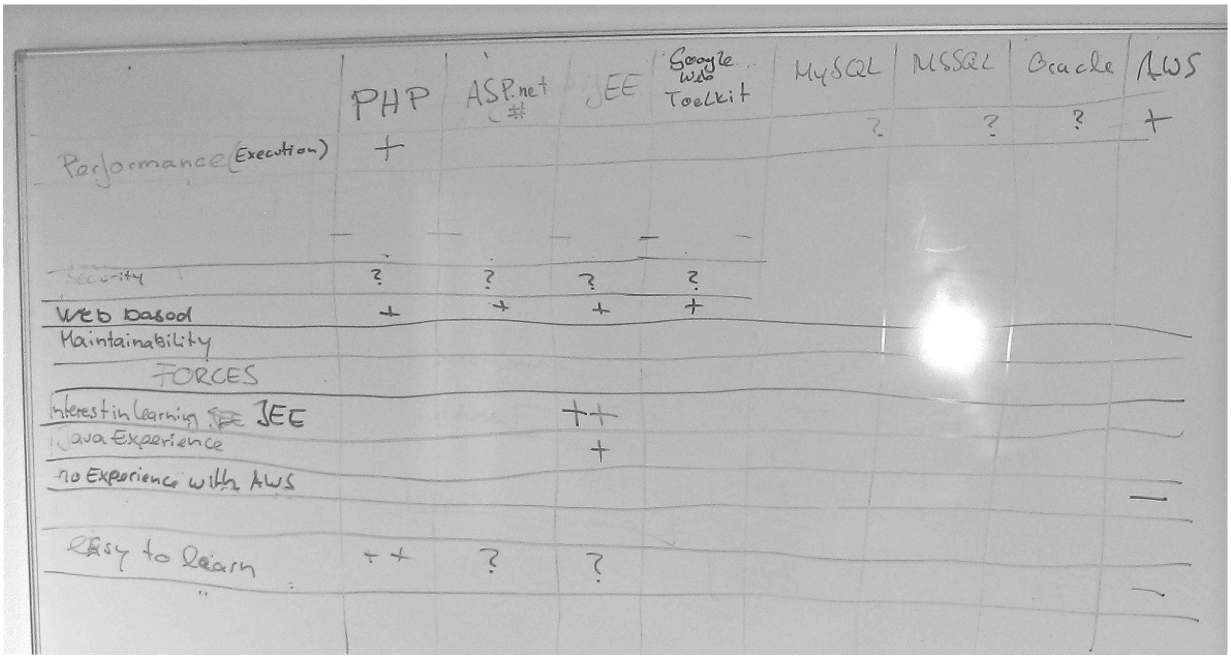


Figure 8.4: *Partial forces view created by project team beta*

Apart from requirements statements, the decision view group also explicitly discussed other decision forces like learnability of technologies or previous experience of the project members. In all cases, the forces view was used as a means to discuss and capture design choices and their arguments.

The decision view project teams systematically filled knowledge gaps that became apparent during the discussions, by conducting research on particular design options (Cat29). Each team member presented the results of the design options research to the other members before a decision was made. This often led to intense discussions, in which the team members heavily challenged each others arguments (Cat30). The students appreciated that decision forces helped to spread knowledge about individual design options better among the different project members and that it provides a framework for discussing options systematically. In the focus groups, both project teams in the decision view group acknowledged that the decision views had a huge impact on the discussions of design options. A member of project team alpha said: "If you don't have the view, then you might also see alternatives, but if you have experience in a solution then you will choose this one. But with the (forces) view, you are forced to think about which one is really better.". In accordance, a member of project team beta stated: "I think the fact that we had to document decisions and our decision making had an influence on the seriousness that we handled decisions. So if there wouldn't have been these views, then maybe we would have been faster in decision making; just say ok that

works, so let's take it".

As opposed to the decision view project teams, the two comparison project teams made decisions mostly without explicit reasoning (Cat31). The project teams searched for viable solutions and applied them without systematically discussing their advantages and disadvantages. When being asked about the rationale behind specific decisions, the students gave the impression that they started thinking about pros and cons just in the moment when the question was asked. In case of project delta, in many cases, the members were not able to provide any rational for major design decisions. Finally, judging from the focus groups and the team minute meetings, the comparison project teams did not discuss design decisions in the group (Cat32). Instead, decisions were made by single members and silently accepted by the other team members.

8.3.7 Resp7 - Discussion of multiple design options in combination

Both project teams in the decision view group used the relationship view to systematically explore technological dependencies before making decisions (Cat33). As mentioned above, the project teams created partial relationship views on the whiteboard to support design discussions. The relationship view showed the students that some decision have a great impact on other decisions. Both project teams spent a lot of effort on understanding these impacts. Many technological decisions were made in combination (Cat34); both project teams not only discussed alternatives for single decisions, but also compared multiple graphs of decisions (multiple combinations of decisions, which as a whole, are alternatives to each other).

The project teams in the comparison group did not discuss multiple design options in combination, but rather made decisions without evaluating impacts on other decisions.

8.3.8 Resp8 - Avoidance of unnecessary complexity.

Only very few indicators were found that any of the student projects explicitly tried to avoid unnecessary complexity in their design. Both of the project teams in the decision view group, however, stated that avoiding unnecessary complexity was an explicit goal within the project (Cat35). When being asked how unnecessary complexity could be avoided, project team alpha stated that they tried to minimize the usage of third party libraries, particularly libraries that come with a lot of unneeded functionality. Project team beta explained that they reduced unnecessary complexity by trying to find a middleware framework that provides a great part of the needed functionality out of the box. The statements of both project teams could be verified by analyzing the forces view and the architectural design; however, no other examples for explicit avoidance of unnecessary complexity could be found.

The project teams in the comparison group did not explicitly avoid unnecessary complexity. However, there is not enough evidence to show that the the use of the decision viewpoints help them to avoid unnecessary complexity.

8.3.9 Resp9 - Validation of design options against the ASRs

As mentioned for Resp4 already, the documentation of requirements was weak in all four projects. However, the decision view project teams at least considered architecture significant requirements when making technological choices using the forces view (Cat36). In the comparison group, no evidence was found that architecture significant requirements were considered when making decisions (Cat37): design decisions were made without systematically identifying alternatives, while there were no indicators that design options were validated against ASRs. The project teams in the decision view group were aware of the fact that in some cases trade-offs between multiple requirements could be necessary (Cat38). Project team alpha used the decision forces view to resolve such situations, but they declared that this happened very rarely (Cat39). Indeed, their forces view showed that the students did not come across many conflicts that had to be resolved.

8.3.10 Resp10 - Prototyping design options

All four project teams heavily used small prototypes to understand technological options (Cat40). In particular, they created prototypes to understand how technologies (e.g. frameworks or libraries) must be used (Cat41). However, only project team beta systematically created prototypes for the purpose of understanding advantages and disadvantages of multiple alternative design options (Cat42). The other projects, in contrast, created prototypes only after a decision was made. Thus, there is no observable influence of the usage of decision views on prototyping.

8.3.11 Resp11 - Evaluation of the architecture as a whole

Apart from the discussion of multiple design options addressing identical problems (e.g. database management systems to be used as a central datastore), none of the four project teams explicitly evaluated the architecture as a whole (Cat44). Nevertheless, when being confronted with this issue, the two project teams in the decision view group mentioned that the decision views allowed them to permanently maintain an overview over the current state of the architecture (Cat43). In particular, they mentioned that the forces view always gave them a good estimate over the coverage of the requirements, that's why they (falsely) assumed that a dedicated architecture evaluation session was not necessary.

8.3.12 Variations of decision view usage

Apart from the findings reported above, we learned that the students in the decision view projects had divergent preferences regarding specific viewpoints. As described in Section 8.2.4, project alpha was a brown-field project, whereas project beta was a green-field. Although the students in project alpha also appreciated the decision forces viewpoint, they saw the most value in the decision relationship viewpoint, because it

helped them to analyze and understand technological dependencies. Taking over an existing software project requires software designers to understand the architecture as-is, before they can make any significant changes. Apparently, the relationship viewpoint helped the students in the brown-field project to analyze and document the system as-is; moreover it helped them to resolve a blocking technical constraint, which had a huge impact on multiple technological design options.

Project team beta, the green-field project, experienced the forces viewpoint as the greatest help in the project, although they also made vast usage of the relationship viewpoint. In green-field projects, the solution space is widely open in the beginning. The decisions made in this project stage are highly important and fundamental to the entire system. The decision forces viewpoint turned out to be a useful support for the students to make decisions based on solid argumentation using an agreed-upon evaluation scheme. It gave them more confidence that the decisions they made were the right ones.

Even though the relationship and forces viewpoints were very well received by the students, both project teams expressed their discontent about documenting the decision detail view. They experienced documenting single decisions using our template as a tedious job that did not have an immediate benefit for the design process. The same finding had been made by other researchers in the past (e.g. (Harrison et al. 2007)). Yet, the students acknowledged that the detail views will have a clear benefit for subsequent developers taking over their project.

8.3.13 Summary of findings

We have found that the decision views provide strong support in the area of solution evaluation and selection, partial support for ASR management, and no support for handling complexity or evaluating the viability of a design option. Table 8.5 summarizes the findings regarding the decision view support (column *Dec. view supp.*) for particular design activities, based on the analysis of the response variables. Decision views provide strong support for design activities related to *architectural synthesis* and *architectural evaluation*[3]. By far the strongest support was recognized for Resp5, related to the discovery of design options (architectural synthesis). The decision views triggered the two project teams in the decision view group to identify multiple options for decision topics and to thoroughly conduct research to understand these options. The project teams in the comparison group, in contrast, clearly did not attempt to identify multiple options before making decisions; they rather chose the first presumably viable solution they could find.

Concerning architectural evaluation, the impact of decision view was significant for Resp6 (balancing advantages and disadvantages of design options). The decision view groups invested much more efforts in researching, understanding, and discussing ad-

[3]Architectural synthesis refers to activities followed to identify candidate architecture solutions for a set of architecture significant requirements; architectural evaluation concerns the validation of those candidate solutions agains all architecture significant requirements (Hofmeister et al. 2007).

Table 8.5: *Summary of findings*

Variable	Variable description	Dec. view supp.
Resp1	Identification of ASRs	+
Resp2	Requirements negotiation	+
Resp3	Prioritization of requirements	+
Resp4	Documentation of requirements	~
Resp5	Discovery of design options	++
Resp6	Balancing advant. and disadvant.	++
Resp7	Discuss mult. options in comb.	++
Resp8	Avoid complexity	~
Resp9	Validate options against ASRs	+
Resp10	Prototyping options	~
Resp11	Evaluation of arch. as a whole	+

++: strong support
+: partial support
~: no support

vantages and disadvantages of the (candidate) architectural solutions than the comparison group, who made decisions mainly implicitly, without discussing them. In addition, the fact that the decision view project teams consciously made multiple decisions in combination (Resp7, architectural evaluation), shows that the decision views stimulated the students to regard dependencies between decisions and contributed to the understanding of the architecture as a whole.

As Table 8.5 shows, the use of decision views did not appear to support the students in (1) requirements documentation, (2) avoidance of unnecessary complexity, and (3) prototyping design options. Point (1) was first a surprising result. The decision forces viewpoint and the decision detail viewpoint explicitly require the statement of requirements, which should have caused the students to define requirements more carefully. A discussion of this finding with the students' lecturers at the university showed that the students were not educated in distinguishing between architecturally relevant requirements and other requirements. They were also particularly inexperienced in documenting quality requirements, and business and environmental demands, which have a higher relevance for architecture decisions. Thus, the fact that none of the project teams documented requirements thoroughly suggests that software designers need to be trained in identifying and documenting architecture significant requirements. Additionally, presenting a checklist of the typical forces in specific domains can remind inexperienced designer in carrying out relevant design activities such as documenting ASRs.

Points (2) and (3) are expected. The viewpoints did not help the decision view group to avoid unnecessary complexity (point (2)). Avoiding complexity obliges designers to simplify and optimize a design solution as far as possible. This requires the knowledge of how a solution can be formulated without compromising the requirements. While the decision views help junior designers to evaluate and select good solutions, they cannot create solution options that are beyond the knowledge of the designers. Prototypes

are a means to evaluate the influence of a design solution on certain qualities of the target system. The decision forces viewpoint can be used to document the results of these evaluations (i.e. the impact of a force on a certain design option) to support a systematic choice out of multiple decision alternatives, but it does not prescribe how the evaluations must be done.

Other activities belonging to architectural analysis (Hofmeister et al. 2007) (Resp1-Resp3) were partially supported by the decision views, mainly because the explicit documentation of decisions and forces on decisions raises a general awareness for aspects that need to be taken into consideration when making decisions.

8.4 Validity

We use the classification scheme proposed by Yin (Yin 2003) and Wohlin et al. (Wohlin et al. 2012) to report on potential threats to validity and measures we took against them.

8.4.1 Construct validity

Construct validity is concerned with the measures used to represent the effect of the method on the study result according to the research conjecture. To ensure validity of the constructs, we identified response variables prior to the study, explained the rationale behind those variables, and assigned them to data collection methods we would use to measure them. Additionally, we used the constant-comparison method to uncover concepts in the qualitative data. The concepts were uncovered from scratch according to their relevance to the study goal, i.e. we had not thought of concepts in advance.

8.4.2 Internal validity

Internal validity is mainly concerned with the examination of cause relationships between the method uses (decision view creation, or ad-hoc) and the response variables. Researchers have to make sure that there are no hidden variables that silently affect the investigated objects. The measures we took to mitigate this risk are two-fold. First we carefully defined case variables that could have an influence on the outcome of the study. Second we used two pairs of projects: one pair that used decision views, and one that did not. The latter reduces the risk of hidden variables. In particular, the following case variables were identified to uncover and deal with potential hidden causalities related to the study results:

- Case variables 2, 3, 4 and 7 concern the previous relevant working experience of the students who took part in the study. To make sure that the experience does not adulterate our results, we made sure that the average experience of the students in the decision view group is at most comparable to the average experience of the

comparison group. In most cases the experience of the students in the decision view group was less than that of the comparison group.

- Case variables 5 and 6 concern characteristics of the software projects. For the validity of the results, it is vital that the projects in the decision view groups are comparable to the projects in the comparison group with respect to the factors under study (i.e. the design process). None of the projects is in a domain that would require an adaptation of the design process (e.g. because of special security or safety needs). Instead, all four projects belong to the software engineering and enterprise computing domains. Project delta was additionally assigned to logistics and web application; project gamma to marketing; projects alpha and beta were both additionally assigned to web application. The difficulty of the projects was balanced among the two groups, as described in Section 8.2.4.

For the evaluation of some response variables (Resp5, Resp6, Resp8-10), we analyzed the decision views created by the decision view groups. This bares a potential risk, as sometimes evidence might only be present in decision views, without being visible in other data collected. In these cases, the cause construct could be under-represented in the comparison group, leading the researchers to wrong conclusions. We mitigated this risk by consulting the decision views only in cases, in which the subjects explicitly mention them in other data collected (e.g. minutes or focus group transcripts). This ensures that no conclusions are drawn based on evidence solely visible in decision views.

8.4.3 External validity

External validity concerns the extent, to which the findings of the study are of relevance for other cases. In this particular case, the study was conducted with students, which is always a threat to external validity, because students are lacking professional experience and real life project constrains like short time-to-market and limited budgets. We partially mitigated the latter issue by using external customers and real software projects. The customers of the projects were independent, i.e. they have no relationship to the school or the researchers. Furthermore, the students were in the last year of a four-year Bachelor of Software Engineering degree, i.e. very close to their professional careers, which is why we assume that the results are at least generalizable to the population of inexperienced software designers with a few years of industrial experience.

Another potential threat to the validity of the results derives from the fact that the students in all projects came from the same university of applied sciences. Theoretically, students from other universities, with a different background, could have performed differently. An identical educational background of the subjects in the two study groups, however, is a prerequisite for the comparison of their design activities. Thus, to completely rule out this potential threat to external validity, the study has to be replicated at other universities. We consider this as future work.

8.4.4 Reliability

Reliability is primarily concerned with the question to what extent the study results are dependent on the specific researchers. We addressed the following issues related to reliability in our study design.

By asking specific questions, the moderator of the focus groups could influence the students towards giving the desired answers (researchers' bias). To mitigate this threat, the moderator asked open questions like "What did you do since our last meeting?" during the focus groups. This starts a discussion between the project members without influencing them. Then the moderator asked the participants to go more into detail or to move on to a different topic. A question guide (Mack et al. 2005) had been prepared in advance to make sure that the students gave enough information to answer the research questions. Question guides help the focus group moderator to focus the discussion on relevant topics. If the discussion deviates from the subjects of interest, he can mildly intervene to put it back on track. The question guide used during the focus groups can be found in D.1.

An additional potential threat to the reliability of the study results could be that students make imprecise, incomplete, or even non-veridical comments during the focus groups. The following measures were taken to mitigate this risk. First, the focus groups were conducted on a weekly basis to make sure that the students' memories were still present. Second, to verify that the students comments correspond to reality, we used methodological and data-source triangulation (Lethbridge et al. 2005). Apart from establishing a broader view of the research object under study, triangulation allows to verify gained impressions using different data-sources and methods. In particular, we were able to check the students' comments by looking into the minutes of their team meetings and the work artifacts they checked in to the Subversion repositories.

8.4.5 Ethical issues

The ethical issues resulting from using students in empirical studies were discussed in Section 8.2.4.

8.5 Related work

Since the late 1980s, researchers have conducted studies to comprehend the design process of software intensive systems (e.g. (Curtis et al. 1988, Sonnentag 1998, Zannier et al. 2007, Brooks 2010, Tang, Aleti, Burge and van Vliet 2010)). The study, presented in this chapter, is related to this research field. In the following, we outline typical design studies in the field and relate them to our own findings. The presented work covers general design studies, studies of decision making in software projects, and studies on the influence of documentation on the design process. To the best of our knowledge, the influence of architecture decision documentation on the design process has not been empirically investigated so far.

In 1988, Curtis et al. interviewed personnel from 17 large software engineering projects to identify the design activities the teams followed (Curtis et al. 1988). The focus of the study was on how requirements were gathered and how design decisions were made and documented, and how these decisions impacted the design process. They identified three common problems among all analyzed projects: 1) Domain knowledge was thinly spread among project members, 2) Requirements were often changing or even conflicting, and 3) Communication and coordination of tasks did not go optimal. Among others, they conclude that staff-wide sharing of (architecture) knowledge has to be facilitated and software development tools should support the representation and management of uncertain design decisions.

Our own findings (almost 25 years later) show that all three identified problems are still perceptible in projects of student software engineers. Using decision viewpoints, however, turned out to at least partially mitigate these problems.

Sonnentag, a German psychologist, analyzed the design process of 40 professional software designers from 16 different software development teams in 1998 (Sonnentag 1998). After the teams solved a predefined design task, she asked each participant to peer evaluate their team mates. This process was used to distinguish high performers from moderate performers (from the perspective of the team mates). In the subsequent analysis phase, she compared the behavior of the high performers and the moderate performers with respect to problem comprehension, planning, feedback processing[4] , task focus, using visualizations, knowledge of software engineering strategies, and length of experience. Her results include that high performers spent twice as much time on feedback processing than moderate performers. She suggests that high performers, who actively evaluate their design solutions, not only perform better at the present task but also gain more experience for future use in other projects. Surprisingly, she also found that the experience of the participants did not have a significant impact on their behavior regarding the previously mentioned aspects.

Sonnentag emphasizes that the repetitive critical evaluation of design options in the context of requirements (and other forces) helps designers to estimate in how far a pursued goal has been achieved. Similarly, our study shows that decision viewpoints, in particular the decision forces viewpoint and the decision relationship viewpoint, provide junior software designers with an organizational structure to support these activities. In another experiment, Tang et al. find that forcing designers to verbalize their design options and reasoning help to bring about better design, especially for junior designers (Tang et al. 2008).

Zannier et al. report on 25 interviews conducted with software designers to develop a model of design decision making in software projects (Zannier et al. 2007). The study focuses on understanding in which situations designers use a rational decision making process versus situations in which the designers follow a naturalistic approach to decision making. Rational decision making, as defined by the authors, is "character-

[4]Sonnentag defines feedback processing as the comparison of a present situation (here the current version of a software design) with the cognitive representation of the design goal at hand. In other words, feedback processing helps the designer to evaluate how far the design goal has been achieved already.

ized by the consequential choice of an option among a set of options, with the goal of selecting the optimal solution", whereas naturalistic decision making is "characterized by situation assessment and the evaluation of a single option with a goal of selecting a satisfactory option". The authors found out that designers generally mix both decision making strategies; however, in cases where the design problem was well-defined, the designers under study primarily used rational decision making, wheres the naturalistic approach was preferred to tackle ill-defined problems.

In our own study, we found out that the subjects in the decision view group followed a more rational decision making process than the subjects in the comparison group, although all four projects had a comparably ill-defined design problem in the beginning. This suggests that the documentation of decision views pushed the students towards structuring the design problem better, in order to identify potential solutions and to define criteria (in the decision viewpoint terminology referred to as *forces*) to choose among the solutions. These findings, however, are not contradictory to those of Zannier et al. As conjectured prior to the study, documenting decision views requires designers to think about design options and evaluation criteria upfront, thus they also implicitly require the user of decision views to structure the design problems at hand.

The same conclusions as Zannier et al. were made by Cross. In a review of multiple empirical studies of design activity in different domains (Cross 2001), Cross acknowledges that designers respond to ill-defined problems by adopting a solution-focused design process. He explains that designers tend to find a *satisfactory* solution rather then systematically generating an *optimal* solution if the problem that needs to be solved is ill-defined. Along with this finding, he states that designers appear to stick to a solution concept as long as possible, even if they encounter shortcomings or difficulties with that solution (this phenomenon is also known as *anchoring* (Epley and Gilovich 2006)). In this study, we also found out that the projects in the comparison group searched for design options until they found a satisfactory solution and sticked to these solutions as long as possible. Assuming that designers, as Cross suggests, are by nature solution-focused and subject to anchoring, the creation of decision views helped the students in the decision view group to alleviate the effects of this phenomenon, by forcing them to consider alternatives and explicitly comparing them in the context of the relevant decision forces.

The assumption that the documentation of design, and the process of designing itself, mutually interfere with each other has also been examined by Purcell and Gero. Purcell and Gero suggest that the majority of cognitive design activities are too complex for all aspects to be held in short-term memory during the design process (Purcell and Gero 1998). They advocate that design sketches can serve designers as external memory device, which can be used to reduce the load on working memory. These findings are in line with the statement of the decision view students that the decision views helped them to maintain an overview over decisions made and over all decision forces that had to be taken into consideration.

Similar studies were conducted by Parnas, who, throughout large parts of his academic career, conducted research on documentation and its importance for the software

engineering process (Hester et al. 1981, Parnas and Clements 1986, Parnas 2009, Parnas 2011). Parnas affirms that software design is a decision making process and that documenting software design "forces designers to make decisions and can help them to make better ones". In (Parnas and Clements 1986), Parnas and Clements emphasize the importance of designers striving to follow a rational design process. In this work, they stressed the need for documentation to record design decisions, ideally guiding the design process of the development team and serving as a reference during software evolution.

8.6 Conclusions

Prior to the study, we conjectured that students would use a more rational design process if they use architecture decision views, compared to students who use an ad-hoc design approach. We characterize a rational design process using eleven response variables. These eleven response variables were used to analyze the design activities that were carried out by the student project teams.

We have found that in three response variables, the decision relationship viewpoint and the decision forces viewpoint have helped students to follow a more rational design process regarding architectural synthesis and evaluation. Students in this group were better at exploring design options, evaluating the advantages and disadvantages of design options and considering the consequences of combining multiple design options.

On the other hand, the viewpoints were ineffective in helping students in three response areas: to manage requirements, to optimize design regarding complexity, and to explore solution viability by means of prototypes. It appears that something more than the use of viewpoints is needed in order to excel in these three response areas.

We suggest that the identification and documentation of architecture significant requirements and other forces should receive more attention in computer science education. Additionally, we plan to investigate if checklists of typical domain-specific requirements and other forces can at least partially fill the gap regarding requirements documentation. Using prototypes for evaluating design options is an important best-practice we identified in our previous work with professional software architects (see Chapter 4). In our opinion, the use of prototypes should be promoted more in higher computer science education; forces can serve as criteria to evaluate design solutions by means of prototypes. Finally, the current set of decision viewpoints cannot support designers in optimizing a software design with respect to complexity. Additional research is needed to identify metrics for complexity that can be used on a decision level, and to subsequently leverage these metrics by means of decision viewpoints.

Acknowledgements

We would like to thank all participating students from the Software Factory course 2011/2012 at the Fontys University of Applied Science in Venlo, the Netherlands.

Part V

Evaluating architecture decisions

In revision for the IEEE Software Magazine as: U. van Heesch, V. Eloranta, K. Koskimies, P. Avgeriou and N. Harrison – *"DCAR - Decision-Centric Architecture Reviews'*

Chapter 9

Decision-centric architecture evaluation

Abstract

Architecture evaluation is an important activity in the software engineering lifecycle that ensures that the architecture satisfies the stakeholders' expectations. Additionally, risks and issues can be uncovered before they cause tremendous costs later in the lifecycle. Unfortunately, architecture evaluation is not regularly practiced in industry.

In this chapter, we present DCAR, an architecture evaluation method that uses architecture decisions as first class entities. DCAR uncovers and evaluates the rationale behind the most important architecture decisions, considering the entire context, in which the decisions were made. Furthermore, it is lightweight and can be performed during or after the design was finalized.

Experiences in large industrial projects have shown that full-scale DCAR evaluations, including reporting, can be conducted in less than five person-days, while producing satisfying results for the stakeholders.

9.1 Introduction

Software architecture that is poorly designed or carelessly cobbled together may cause an entire software project to fail. Therefore, it is important to evaluate software architecture early on in the development. Various software architecture evaluation methods have been proposed to uncover architectural problems in a systematic way (Dobrica and Niemela 2002, Bass and Nord 2012). The most popular evaluation methods are scenario-based, e.g. ATAM (Kazman et al. 2000). In general, architecture evaluation has several benefits. Most importantly, a problem or a risk identified early in the development process can be easily fixed or mitigated, compared to a problem that is found late, e.g. in the testing or integration phase, or even during maintenance. Furthermore, architecture evaluations encourage communication among the involved stakeholders that would not take place otherwise.

Despite these benefits, architecture evaluation is not regularly practiced in the industry today (Dobrica and Niemela 2002, Bass and Nord 2012). A study with software architects uncovered typical factors that influence the architecture evaluation practices of organizations (Babar et al. 2007). According to the study, management commitment, company-wide evaluation standards, a funding model, and appropriate training for the involved staff members are prerequisites for establishing architecture evaluations in a

company. Such prerequisites are often not met. Furthermore, the increasingly popular agile development approaches do not encourage the use of architecture evaluation methods, which often consume a considerable amount of time and resources.

In order to lower the threshold of industrial adoption of architecture evaluations, we developed a new evaluation method called DCAR (Decision-Centric Architecture Review). DCAR was developed bottom-up, based on our experience on performing architecture evaluations in the industry and observing what works well in practice. This lead to two high-level requirements. First, the method needs to be light-weight in terms of required time and resources. Second, the method must support the evaluation of software architecture decision-by-decision, allowing systematic analysis and recording of the rationale behind architecture decisions. The latter requirement makes the method different from scenario-based methods, which aim at testing a software architecture against scenarios that refine the major quality requirements of the system.

DCAR is decision-centric in the sense that the evaluation is carried out by selecting a set of decisions which are analyzed in the context of all relevant project- and company-specific decision forces. A decision force, or force in short, is any non-trivial influence on an architect who is looking for a solution to an architectural problem. The concept of forces is elaborated below. DCAR can be used for any set of architectural decisions, of any type. Its is applicable for all types of software-intensive systems and domains.

We have carried out multiple DCAR evaluations in the industry. Our experience indicates that an average DCAR session takes half a day, requiring the presence of 3-5 members of the project team, including the chief architect. Thus, the total amount of company working hours is less than four person-days plus another two person-days for the review team. This makes DCAR suitable for projects that do not have the budget, schedule, or stakeholders available for full-fledged architectural evaluations. DCAR is also pertinent for projects that wish to perform architecture evaluation to justify a set of architecture decisions rather than for ensuring that a whole system satisfies its quality requirements. Table 9.1 presents a short profile of DCAR, using the classification proposed by Bass and Nord (Bass and Nord 2012).

9.2 Architecture Decisions

DCAR focuses on evaluating specific architecture decisions, selected by stakeholders under the assistance of the review team. Understanding architecture decisions and the rationale behind them is crucial for continuously ensuring the integrity of a system.

Architecture decisions are the fundamental choices, an architect has to make, concerning the overall structure or externally visible properties of a software system (Tyree and Akerman 2005). Typical examples of such decisions are the choice of an architectural pattern or style, the selection of a middleware framework, or the decision not to use open source components for licensing considerations.

Architecture decisions are not isolated; they can be seen as a web of interrelated decisions that can depend on, support, or contradict each other. Some decisions must be

Table 9.1: *DCAR short profile*

Context factor	**DCAR**
Inputs	Informal description of requirements, business drivers and architectural design; generated prior to the review.
Output	Risks, issues, and a thorough documentation of the evaluated decisions and their decision forces.
Reviewers	Company-internal or external reviewers (preferred).
Priority setting of decisions	During the review.
Social interaction	Face to face meeting.
Resources required	2-4 reviewers, architect, developers, business representative.
Skill level of participants	Moderate.
Knowledge of evaluators	General architecture.
Tools or automation	Templates, Wiki, UML tool.
Evaluation objectives	Rationale behind decisions.
Scope	A set of specific architecture decisions.
Schedule	Half a day preparation and post processing, half a day review session.
Project phase	Within or after the architectural design is finalized.

combined to achieve a desired property; other decisions are solely made to compensate the negative impact on a desired property, caused by a decision made to achieve a different property. As an example, an architect could decide to use an in-memory database to achieve short response times. This decision has a negative impact on reliability, which, in addition to short response times, is another desired property of the system. To compensate this negative impact, the architect could decide to use redundant power supplies, or to replicate the database and the hardware and use the replica as a hot spare. Therefore, the decisions to use redundant power supply and a hot spare would be caused by the decision to use an in-memory database.

In DCAR, the participants identify the architecture decisions made and clarify their interrelationships. This is primarily done for two reasons: First, understanding the relationships helps to identify influential decisions that have wide-ranging consequences for large parts of the architecture. Second, when a specific decision is evaluated, it is important to consider its related decisions as well (as illustrated by the previous *in-memory database* example).

9.2.1 Decision forces

Several factors need to be taken into consideration to evaluate an architecture decision. Apart from architecturally-significant functional and non-functional requirements, such

factors include constraints, risks, political or organizational considerations, personal preference or experience of the architect and the development team, or business goals like quick time-to-market and low price. We call these factors *decision forces* (see Chapter 7), because of the similarities with forces in physics. Each force has a direction and a magnitude. It either pushes an architect towards a specific solution, or it pushes the architect away from that solution.

In order to evaluate an architectural solution, the related decisions also need to be contemplated and considered as decision forces. In their totality, forces reveal the entire context in which a decision is made. As some of them can be conflicting, or orthogonal to each other, an architect has to balance all forces to make the best possible decision. Figure 9.1 illustrates the concept of forces using the in-memory database decision, introduced above. In this particular case, the forces in favor of the in-memory database outweigh the forces against it. DCAR explores the entire rationale behind decisions by

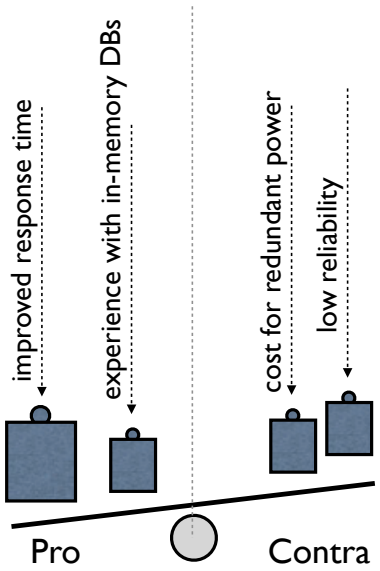


Figure 9.1: *Some of the forces for the in-memory database decision*

means of forces. After identifying forces, the review participants examine if the rationale behind the evaluated decision is still valid in the current context. This is important, because forces are not immutable; not only requirements keep changing, but the tactical orientation of the company may evolve: laws and regulations may have changed, or new technologies could exist that would offer a better solution to a design problem at hand. So these changes in the design context may change the magnitude of the forces, or even introduce new forces and make some of the old forces obsolete. In the new design context, if the negative forces outweigh the positive forces, then the reviewers recommend to reconsider the decision.

9.3 Introducing DCAR

9.3.1 Company participants and review team

To achieve best results, DCAR requires the participation of the lead architect and one or two members from the development team with different roles and responsibilities in the software project. Additionally, somebody has to represent the management and customer perspective. This is important, because some decisions have to be assessed from an enterprise-wide perspective rather than taking only project-specific forces into account.

The review can be done by external reviewers, or by an organization's own people who are not involved in the project under review. The members of the review team need to have experience in designing software architecture; ideally, but not necessarily, in the same domain as the system under review.

9.3.2 Essential steps

In this subsection, we briefly[1] present how DCAR is carried out in practice.

Figure 9.2 shows the main steps of DCAR, as well as the produced artifacts (boxes on the right). In the following, each of the steps is briefly described. Step 1 is carried out offline, all other steps are carried out during one evaluation session, in which all participants gather in one room.

Step 1) A date for the DCAR session is settled, and the stakeholders are invited to participate. The lead architect of the system[2] is asked to prepare a presentation of the architecture. This presentation should contain the most important architectural requirements, high-level views on the architecture, used architectural approaches like patterns or styles, and the used technologies like database management systems or middleware servers. The representative of the management and customer perspective is asked to prepare a presentation describing the software product and its domain, the business environment, market differentiators, and driving business requirements and constraints. Templates for both presentations can be found on the previously mentioned DCAR website.

The slides for the presentations are sent to the review team prior to the evaluation session, so that they can prepare for the evaluation. In particular, the reviewers study the material to elicit potential architecture decisions and decision forces. Additional system documentation is not mandatory, but any additional material that can be used by the reviewers to understand the system upfront is helpful.

Step 2) The evaluation session starts with an introductory presentation of the DCAR method to all participants. This includes the schedule of the day, introduction of the

[1] A more elaborate description of the method can be found online at `http://www.dcar-evaluation.com`

[2] DCAR cannot only be used to evaluate whole systems, but also for major and minor sub-systems. For the sake of simplicity, we refer to all of them as *systems*.

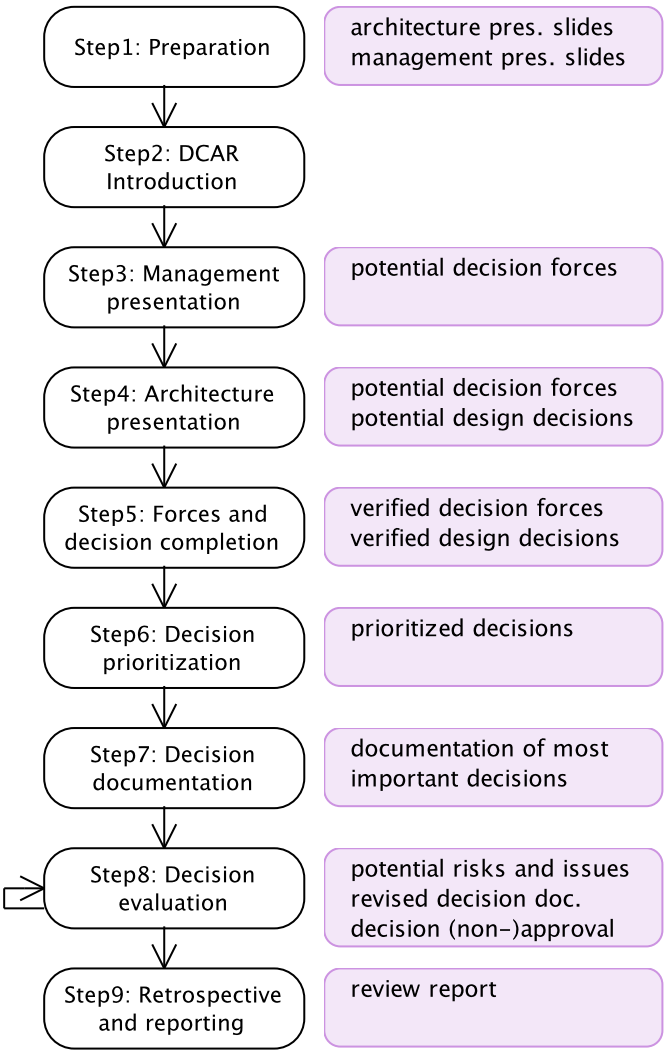


Figure 9.2: *DCAR steps and produced artifacts during each step*

DCAR steps, the scope of the evaluation, possible outcomes, and the roles and responsibilities of all participants. The DCAR website provides an example of such a presentation.

Step 3) The representative of the management and customer perspective gives a short presentation using the slides prepared in Step 1. In our experience, 15-20 minutes suffice, but more time can be used if the schedule allows it. The main purpose of this step is to allow the reviewers to elicit business-related decision forces that must be taken into consideration during the evaluation. The review team notes down potential forces during the presentations and asks questions to elicit additional forces. The management representative does not need to be present during the rest of the session; however, it is beneficial as he or she can provide additional insights during the decision analysis.

Step 4) The lead architect uses the slides prepared in Step 1 to introduce the architecture to all DCAR participants. In our own industrial DCAR sessions, we reserved between 45 and 60 minutes for this presentation. The goal is to give all participants a good mental picture of the architecture. It is supposed to be highly interactive. The

review team and the other participants ask questions to complete and verify their understanding of the system. During this step, the reviewers revise and complete the list of architecture decisions they had identified as a preparation in Step 1. Identifying architecture decisions requires some experience. As a starting point, reviewers can focus on used technologies like servers, frameworks, or third-party libraries. Additionally, it has been a good practice to search for applied patterns in the architecture (Harrison and Avgeriou 2011).

Apart from capturing architecture decisions, the reviewers revise and complete the list of forces they had identified in Steps 1 and 2. Forces can be documented as informal statements. Both, decisions and forces, are revisited in the next step.

Step 5) At this stage, the reviewers have assembled a preliminary list of architecture decisions and decision forces. The goal of Step 5 is two-fold: Clarify the architecture decisions and their relationships, and complete and verify the forces relevant to these decisions. To support the clarification of the decision relationships, a decision relation-

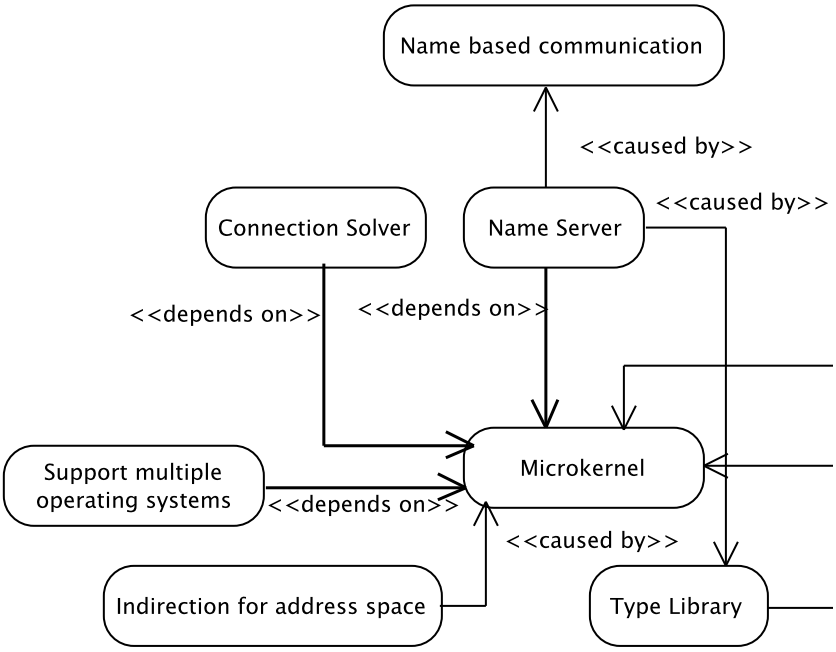


Figure 9.3: *Excerpt from a relationship view created in a DCAR session.*

ship view (see Chapter 6) is created by one of the reviewers during the review session. The diagram is constantly revised during the previous steps. Figure 9.3 shows an excerpt of such a diagram. Each decision is represented by an ellipse that contains a short descriptive name for the decision. It is important to use the company's own vocabulary in the names, to make sure that reviewers and company stakeholders have the same understanding of the applied architectural solution. In the beginning, each of the decisions collected by the reviewers in the previous step is represented in the diagram. After all participants gained a collective understanding of the decisions, the relationships are established. In a relationship view, they are represented by a directed line. Although more relationship types exist (all available types are defined in Appendix B.3.2), in an

architecture review, the only important relationships are *caused by* and *depends on*. The relationships help the reviewers and stakeholders to estimate the importance of each decision. Decisions being the origin of many dependencies have a central role and must be seen critically. Relationships are also helpful to understand which decisions must be taken into consideration as decision forces for other decisions. Relationship views can be created using any UML tool. A template for such a diagram can be downloaded from the DCAR website.

The forces are presented as a bullet list. They should be formulated unambiguously using domain specific vocabulary. Example forces from the machine control domain are *Firmware level design and implementation should be sourced out, as it is not our core business.*, or *We have a lot of in-house experience with the CANOpen protocol*. The review team discusses and completes the list of forces with the company participants.

Step 6) Usually, the number of decisions elicited in the previous steps is too large to discuss each of them during the review. Therefore, the stakeholders have to negotiate which decisions will be reviewed in the following steps. The criteria for selecting which decisions will be reviewed is context dependent and has to be negotiated between the stakeholders. The criteria could be mission-critical decisions, decisions known to bear risks, or decisions causing high costs, for instance.

We use the following procedure to prioritize decisions: Each participant gets 100 points. The points can be distributed freely over the decisions, based on the previously agreed criteria for the importance of decisions. Then the points of all participants are summed up and the the rationale behind each person's ratings is discussed. The decisions with the highest ratings are taken to the next steps. In our experience, the number of decisions that can be discussed effectively in half a day is seven to ten.

Step 7) The architect and the other company participants document the set of decisions that received the highest ratings in the previous step. Therefore, each of them selects two or three decisions he or she is knowledgeable about. The decisions are documented by describing the applied architectural solution, the problem or issue it solves, known alternative solutions, and the forces that need to be considered to evaluate the decision. The stakeholders use the list of forces assembled in the previous steps to make sure that they don't forget important ones, but they can also think of new forces. A member of the review team supports the stakeholders during this process. This is particularly important, because some of the participants may be unexperienced in decision documentation.

Figure 9.4 shows an example of the decision documentation template used in DCAR; other established templates could be used alternatively, e.g. the template defined by (Tyree and Akerman 2005) or (Harrison and Avgeriou 2011).

Step 8) The documented decisions (seven to ten on average) are subsequently evaluated, starting with the decision with the highest priority. The participant, who documented the current decision presents it briefly. After that, the company participants together with the reviewers challenge the decision by identifying additional forces against the chosen solution. They use the elicited decision forces and the decision relationship view to understand the context of the decision, i.e. related decisions and relevant forces

Name	Redundancy of controllers			
Problem	The application should run even if the server fails			
Solution or description of decision	The system is deployed to two servers: one is active, the other one is inactive. The active server provides all system services, while the passive one is running in the background. When the active server fails, the inactive server becomes active. During the switch over, the active server tries to update the passive one to make sure that it has the same data and status. Both servers have an identical software configuration. This solution follows the *Redundant Functionality Pattern*.			
Considered alternative solutions	Apply the *Redundancy Switch Pattern*: Both servers are active; external logic is used to decide which output is actually used in the control. In this case, cyclic data copying could be avoided. However, applying this solution would require major modifications to the system. Even though availability would be increased, it would also cause additional costs. The customers are not prepared for paying more for higher availability. Additionally, the external logic component could become a potential single point of failure. Therefore, this alternative was discarded.			
Forces in favor of decision	• Easier to implement than the alternative solution. • Scales easily to versions where redundancy is not used. • No additional costs			
Forces against the decision	• Slower switch over time than the alternative would have. • Hard to offer higher availability than the current 99.99%			
Outcome	Green	Yellow	Yellow	Red
Rationale for outcome	Current solution seems to be ok.	I am concerned about the slow switch over time.	Widely accepted solution. Availability might become a problem in the future	We should really reconsider this decision, as the next release is likely to have higher availability requirements.

Figure 9.4: *Example of documented and analyzed decision*

in favor or against the applied solution. The documentation of the decisions and the decision relationship view are continuously updated by one of the reviewers during this step. All participants discuss whether the forces in favor of the decision outweigh the forces against it.

Finally, all participants decide in a voting procedure, whether the decision is good, acceptable, or if it has to be reconsidered. Figure 9.4 shows the result of an evaluated decision, created during a DCAR session. The traffic light colors indicate the ratings of all participants; green for *good*, yellow for *acceptable*, and red for *has to be reconsidered*. Additionally, it shows justifications for the votes, given by each voter ("Rationale for outcome").

During the whole discussion, the reviewers note down potential issues or risks that were mentioned. Each decision is discussed for approximately 15-20 minutes. In our experience, the quality of discussion diminishes at some point. If a decision requires more than 20 minutes, it can be flagged as a point for future analysis.

Step 9) After all of the selected decisions were evaluated, he review team collects all notes and artifacts created during the evaluation session. They serve as input for the

evaluation report. The review team writes a report within two weeks from the review session. The report is discussed with the architect for verification and eventually refined by the review team. In our own DCAR sessions, the report was prepared by the review team on the next day. The advantage of an early report is that the review team and the architect can still vividly remember the discussions held during the evaluation session.

9.4 Experiences

DCAR has been developed in cooperation with industrial partners from the distributed control system domain. As described in Section 9.1, however, it is by no means restricted to this domain. Since the initial version, DCAR has been applied and refined in five large software projects. In this section, we report on our findings from three industrial DCAR sessions, which were conducted in different projects at Metso Automation in Tampere, Finland.

Table 9.2: *Descriptive statistics*

Variable	**Value**
Avg. system size	600 000 SLOC
Avg. no of elicited decisions after step 5	21 decisions
Avg. no of decisions documented in step 7	9 decisions
Avg. no of decisions evaluated in step 8	7 decisions
Avg. no of reviewers	4 persons
Avg. no of company stakeholders	4 persons
Avg. effort for reviewer team	50 person-hours
Avg. effort for company stakeholders	23 person-hours

Table 9.2 shows descriptive statistics about the DCAR executions. The systems under study in these evaluations came from the process automation domain; each system had been in use for multiple years. Each of the three DCAR sessions, summarized here, was conducted within five hours. The effort that the company stakeholders had to spent on the reviews reveals the time spent by the participants for preparation, taking part in the evaluation sessions, and reviewing the evaluation report.

To gather feedback on the participants' perception of DCAR, we carried out interviews with a subset of the participants, including the chief architect. Apart from the chief architect, who naturally knows the architecture best, all interviewees mentioned that they got a good overview of the system's architecture; something they were missing in their daily work, because they were only responsible for smaller sub-systems. They also stated that they liked that all important decisions, even if they were considered stable, were brought into question for the purpose of the evaluation. The prioritization procedure in DCAR step 6 made sure that bias on behalf of the decision maker or the responsible architect was reduced. Systematically discussing decisions in a group also helped them to understand different points of view that need to be considered in the

context of a decision.

Generally, the stakeholders reported that the interaction between the stakeholders, and the discussion with the review team as external contributors, was one of the most valuable advantages of the evaluation session. The chief architect noted that the evaluation report, produced by the review team, is a valuable supplement to the existing system documentation. The interviewees estimated that the decisions elicited during the evaluation roughly covered the most important 75% of all significant architecture decisions; this was regarded as an excellent result given the short amount of time invested in the evaluation.

The success of a DCAR depends on the stakeholders' understanding of the concepts of architecture decisions and decision forces. Therefore, we explicitly addressed these issues in the interviews. Although all interviewees were either already familiar with both terms, or grasped the concepts quickly during the DCAR introduction in Step 2, some of them mentioned that the time given for the documentation of decisions in Step 7 was too short. This was particularly the case for stakeholders, who had never systematically documented architecture decisions before. They proposed to tackle this problem by providing examples of documented decisions prior to the evaluation.

During the evaluations, we observed that the documentation of reasoning, i.e. forces in favor or against a specific solution, was especially challenging for some of the participants. Therefore, in the later evaluations, we provided decision examples with a list of typical decision forces in the domain at hand and found out that it alleviates the problem.

These positive experiences in the past evaluations, and the continuous interest of other industrial partners to hold more evaluations in the future, show that DCAR helps organizations to adopt architectural evaluations as part of their practices. Additional empirical studies will be conducted to provide evidence concerning how far DCAR indeed lowers the threshold for industrial adoption of architecture evaluations.

9.5 Conclusions and Future Work

In this chapter, we introduced DCAR, a decision-centric method for performing architecture evaluations. The method has proven to work in industrial projects and the stakeholders' feedback is promising and encouraging.

DCAR can be carried out in less than a working day and takes the entire decision-making context into account to understand the complete rationale behind decisions. DCAR can be conducted before or after the architectural design was finalized. In addition, DCAR can be repeated, whenever there is substantial amount of new architectural decisions made. We are currently evaluating, how DCAR can be used incrementally in the context of agile software projects. Furthermore, tool support for documenting decisions and for creating decision relationship and decision forces views will be offered.

Part VI

Conclusions and future work

Chapter 10

Conclusions and future work

This chapter concludes the dissertation. At first, I revisit the research questions defined in Chapter 2.3, summarize the answers to the research questions, and list the contributions of this research project. Subsequently, I discuss directions for ongoing and future work.

10.1 Answers to research questions and contributions

Chapter 2 introduced the main problem statement, which formed the basis for the work presented in this dissertation: "Existing architecture decision modeling approaches do not satisfy all stakeholders' concerns, they do not integrate with viewpoint-based architecture descriptions, and they do not optimally support architectural analysis, architectural synthesis, and architectural evaluation."

Multiple research questions were derived from this central problem, and addressed in Chapters 3 to 9. Table 10.1 summarizes the research questions, the chapters in which they were addressed, and the contributions made with respect to the research questions. In the following, each research question is revisited and briefly discussed.

RQ 1: *How are ADs made?*

Before looking into ways on how to improve decision modeling approaches in order to address the central research problem, we first had to understand how decisions are really made in practice (RQ 1). To answer RQ 1, we carried out two surveys. The first survey was conducted with final- year software engineering students prior to graduation. We learned how inexperienced software designers make decisions and, by comparing these results to architecting approaches in the literature, identified areas in the students' reasoning processes that need to be improved (see Chapter 3).

The second survey was conducted with software architects from the industry. The goal of this study was to explore the decision-making process of professionals. As a result, we identified several reasoning best practices that can be studied by inexperienced architects to optimize their decision-making process (see Chapter 4).

RQ 2: *How to improve the way ADs can be modeled?*

To make architecture decision modeling more efficient, we first thought about an approach to decision documentation that requires less effort during the architecting process than template-based decision documentation, e.g. using the template

Table 10.1: *Contributions*

	Research questions	Chapter	Contribution	Explanation
Understanding ADs	RQ 1: How are ADs made?	Chapter 3	Finding: Areas for improvement in the reasoning process.	Areas for improvement in the reasoning process, derived from a survey with senior software engineering students.
		Chapter 4	Finding: Reasoning best practices.	A set of architectural design reasoning best practices collected from professional software architects.
Modeling ADs	RQ 2: How to improve the way ADs can be modeled?	Chapter 5	Finding: A focus on patterns during AD recovery leads to higher quality of recovered ADs, but not to higher quantity.	A focus on identifying applied patterns in an architecture leads to higher quality of recovered decisions, because the rationale for applying patterns is known and can be reused to document the decision to apply the pattern.
		Chapter 6	A documentation framework for ADs.	Four viewpoints for documenting ADs using the conventions of ISO/IEC/IEEE 42010.
	RQ 3: How to extend the AD modeling approach to satisfy concerns related to decision-force traceability?	Chapter 7	Decision Forces Viewpoint.	An additional viewpoint for the framework dedicated to decision-force traceability.
Supporting ADs	RQ 4: Does modeling ADs using viewpoints lead to more rational ADs?	Chapter 8	Finding: AD documentation supports inexperienced software engineers in following a rational design process.	Students using the decision framework follow a more rational design process regarding architectural analysis and synthesis.
Reviewing ADs	RQ 5: How to support architecture evaluation using AD models?	Chapter 9	A method for decision-centric architecture evaluation.	DCAR is an architecture evaluation approach that uses architecture decisions as primary evaluation targets. It makes use of forces to capture and evaluate the rationale behind decisions.

proposed by Tyree and Akerman (Tyree and Akerman 2005). The idea was to recover decisions after the fact, by identifying applied architectural patterns in the architecture. Finding ways to recover decisions is also particularly important, because architecture decisions were often not documented in the past and the people who made the decisions are frequently not available for inquiry anymore. We hypothesized that, compared to decision recovery with no particular focus, architecture decision recovery is more efficient, if it focuses on identifying applied architecture patterns. We conducted a controlled experiment with academics and practitioners to test this hypothesis (see Chapter 5). The results of the experiment show that a focus on patterns leads to higher quality of recovered decisions, but not necessarily to higher quantity.

After finding out that patterns can be used to recover and describe architecture decisions effectively, we started dealing with the problem how decision modeling approaches can be integrated with viewpoint-based architecture description, while satisfying all stakeholder concerns in decision documentation. Therefore, we developed a framework for architecture decisions, following the conventions of ISO/IEC/IEEE 42010, the international standard for architecture description (Chapter 6). The framework includes four viewpoints, a decision relationship viewpoint, a decision chronology viewpoint, a decision stakeholder involvement viewpoint, and a decision detail viewpoint. Each of the viewpoints was designed to satisfy specific stakeholder concerns in architecture decisions, which were identified by means of a literature survey prior to the development of the viewpoints (see Appendix B.1). With the exception of the stakeholder involvement viewpoint, the framework was evaluated in an industrial case study (see Section 6.4). The results show that decision views can be created with reasonable effort while addressing many decision-related concerns. The conformance to ISO/IEC/IEEE 42010 allows to combine decision views with other viewpoint-based architecture descriptions.

RQ 3: *How to extend the AD modeling approach to satisfy concerns related to decision requirements traceability?*

The decision framework, which was developed as a result of RQ2, did not sufficiently address concerns related to the traceability between architecture decisions and architecturally significant requirements. Architecturally significant requirements are a subset of all forces that influence architects when making decisions. Therefore, instead of limiting the AD modeling approach to architecture significant requirements, we decided to develop a solution that would allow for traceability between architecture decisions and all types of decision forces.

As a result, we designed the decision forces viewpoint, which integrates with the previously developed framework for architecture decisions and satisfies concerns related to traceability between architecture decisions and all kinds of decision forces (Chapter 7). The viewpoint was validated in a multiple-case study with final-year software engineering students working in non- academic software projects (Section 7.4). The study provided evidence that the viewpoint satisfies decision-forces traceability concerns, while being well received by the study participants.

RQ 4: *Does modeling ADs using viewpoints lead to more rational ADs?*

The decision framework, and the forces viewpoint as an extension to the framework, satisfy typical stakeholder concerns in decision documentation and integrate with viewpoint-based architecture description. In RQ 4 and RQ 5, we investigated if decision viewpoints provide support for architectural analysis, synthesis, and evaluation. RQ 4 is primarily concerned with architectural analysis and synthesis, while RQ 5 concerns architectural evaluation.

To answer RQ 4, we conducted a comparative multiple-case study with four groups of software engineering students in the final year of their studies (Chapter 8). Half of the student groups used the decision detail viewpoint, the decision relationship viewpoint, and the decision forces viewpoint from our framework to model their decisions. We selected these three viewpoints, because we considered them particularly helpful for supporting architectural analysis and synthesis, while the other viewpoints mainly serve documentation purposes.

We found out that particularly the relationship viewpoint and the decision forces viewpoint provide strong support in the area of candidate solution evaluation and selection, partial support for the management of architecturally significant requirements, but no support for handling complexity or evaluating the viability of a design option.

RQ 5: *How to support architecture evaluation using AD models?*

RQ 5 concerns the support of decision viewpoints for architecture evaluation. To address RQ 5, we developed an architecture evaluation method that uses decision viewpoints, particularly the decision relationship viewpoint and the forces viewpoint, to support the evaluation process. In Chapter 9, we report on the method called *decision-centric architecture review* (DCAR). DCAR is lightweight and it can be performed during or after the design was finalized. In DCAR, architecture decisions are evaluated taking all important forces of the stakeholders into account. The goal is to understand the complete rationale behind decisions, before judging their suitability and appropriateness in an architecture.

DCAR was developed in cooperation with Finish companies from the machine-control system domain. First empirical evidence for the effectiveness and applicability of DCAR was collected in interviews conducted with DCAR participants after the evaluations of large industrial software systems.

10.2 Ongoing and future work

As suggested by the title of this dissertation, the work presented here can be seen as a next step towards exploiting the full potential of explicit architecture decisions. In this section, I outline areas for ongoing and future work remaining after this research project. The section is divided according to the four parts in the main body of this dissertation: understanding ADs, modeling ADs, supporting ADs, and evaluating ADs.

10.2.1 Understanding architecture decisions

We conducted studies with inexperienced software engineers on the one hand and professional architects on the other hand, to understand their decision-making processes. Additionally to the work reported in this dissertation, we carried out another large scale

questionnaire-based survey with software engineers from the industry and from universities across Europe, who have different years of industrial experience and different levels of education (university, or university of applied science). The goal of this study was to understand the influence of working experience and educational background on the way decisions are made. A preliminary evaluation of the study results showed that the educational background has no significant impact, but experience seems to have a major influence on the decision-making process. The more experience, the more attention is payed to prioritizing requirements, comprehensive exploration of the solution space, reflection on pros and cons of candidate solutions, and understanding and regarding dependencies between architecture decisions. The analysis and interpretation of the study results are still in progress.

10.2.2 Modeling architecture decisions

The framework for architecture decisions, consisting of the five viewpoints described in Chapters 6 and 7, is a self-contained framework, which can be used out-of-the-box to model architecture decisions in different ways to satisfy different decision-related concerns. One concern, however, is not entirely addressed by the viewpoints in the framework: *What decisions influence decision D, or architecture element E?* (Concern C12 in Table 6.1). The first part of the concern, about the mutual influence of decisions, is covered by the decision relationship viewpoint; the second part about the influence of an architecture decision on other architectural elements (e.g. components and connectors) is not addressed by our viewpoints. In order to achieve this type of traceability between architecture decisions and other architectural elements, decision views have to be combined with other architectural views, e.g. the ones described by Clements et al. (Clements et al. 2010). The conformance to ISO/IEC/IEEE 42010, in principle allows to integrate decision viewpoints with such viewpoints. We plan to investigate how the set of viewpoints, we proposed, can be specifically integrated with other viewpoints, in order to achieve traceability and maintain consistency between decisions and other architectural description elements. Additionally, we are currently investigating possibilities to store information about related design decisions directly in source code.

Another important future work area, related to decision modeling, is the establishment of a tool, or a tool chain that supports the creation of decision views according to our viewpoint definitions. Throughout the entire research project, we have been developing a prototype implementation of such a tool, called *Open Decision Repository* (please refer to Section 6.6 for more details). Currently, the Open Decision Repository does not support the decision stakeholder involvement viewpoint and it only partially supports the decision forces viewpoint. We plan to integrate these viewpoints into the tool and validate it in industrial studies.

10.2.3 Supporting architecture decisions

As described in Section 10.1, creating decision views according to our framework only partially supports a rational decision-making process. We see additional potential for improving the reasoning process of particularly inexperienced software engineers, by creating practice-oriented guidelines for decision-making from the best practices, we identified in Section 4.6. These guidelines could for instance be formulated using the pattern format (see (Buschmann et al. 2007) for a description of patterns and pattern languages), which is a well-known and accessible format for software engineering practitioners.

Additionally, as suggested in Section 8, we plan to collect decision forces typical for particular domains. Collecting typical domain-specific forces can be beneficial in multiple ways: first, they can support architects in identifying the forces that are relevant to specific architectural problems; second, they can reduce the effort and maximize the quality of the documentation of forces, because a part of the forces does not have to be reformulated for each individual software project. Architects can reference the typical domain-specific forces in the system-specific forces documentation.

10.2.4 Evaluating architecture decisions

DCAR is a promising approach to lightweight and efficient architecture evaluation. Compared to other architecture evaluation methods, we consider DCAR as particularly suitable for agile software development, because it allows for incremental architecture evaluation. We plan to conduct empirical studies to further explore and validate the suitability of DCAR in agile projects. Furthermore, we are optimizing DCAR for the use in distributed software projects with multiple industrial partners involved in a project.

Appendix A

Appendix to Chapter 5

A.1 Raw data - quality ratings and decision types

Table A.1 lists all decision documented by the participants of the experiment. The first column denotes the execution group, i.e., the execution at the EuroPLoP conference, or the execution at the software architecture workshop. Column two refers to the experimental group. The column called *Part.* contains the number of the participant, who documented the decision. The next column contains a unique identifier of the decision. The last three column contain the results of the quality ratings given by the two analysts (column Al.1, and Al.2), as well the calculated average of these two ratings (column Avg.).

Table A.1: Quality ratings and decision types

Execution Grp.	Group	Part.	Dec.No	Type	Al. 1	Al. 2	Avg.
EuroPLoP	Control	51	15639	Pattern	3	1	2
EuroPLoP	Control	51	31219	Other	1	2	1.5
EuroPLoP	Control	51	35295	Other	2	1	1.5
EuroPLoP	Control	51	57732	Other	1	1	1
EuroPLoP	Control	51	58823	Pattern	3	1	2
EuroPLoP	Control	52	19704	Other	1	1	1
EuroPLoP	Control	52	21539	Other	1	1	1
EuroPLoP	Control	52	23027	Other	1	1	1
EuroPLoP	Control	52	24025	Other	3	1	2
EuroPLoP	Control	52	28014	Other	3	2	2.5
EuroPLoP	Control	52	29573	Other	1	1	1
EuroPLoP	Control	52	30292	Other	2	1	1.5
EuroPLoP	Control	52	33148	Other	3	1	2
EuroPLoP	Control	52	41748	Other	1	1	1
EuroPLoP	Control	52	42404	Other	2	1	1.5
EuroPLoP	Control	52	45167	Other	3	1	2
EuroPLoP	Control	52	51001	Other	2	1	1.5
EuroPLoP	Control	52	51331	Pattern	2	1	1.5
EuroPLoP	Control	52	55931	Other	3	1	2
EuroPLoP	Control	52	56835	Other	3	1	2
EuroPLoP	Control	52	63345	Other	1	1	1
EuroPLoP	Control	52	63653	Other	1	2	1.5
EuroPLoP	Control	52	69428	Pattern	2	1	1.5
EuroPLoP	Control	52	76435	Other	2	1	1.5
EuroPLoP	Control	53	15154	Pattern	5	3	4
EuroPLoP	Control	53	23289	Pattern	4	2	3
EuroPLoP	Control	53	25919	Pattern	5	2	3.5
EuroPLoP	Control	53	35942	Other	5	2	3.5
EuroPLoP	Control	53	41417	Pattern	5	2	3.5
EuroPLoP	Control	53	46622	Pattern	4	3	3.5

Table A.1 – continued from previous page

Execution Grp.	**Group**	**Part.**	**Dec.No**	**Type**	**Al. 1**	**Al. 2**	**Avg.**
EuroPLoP	Control	53	51204	Pattern	5	2	3.5
EuroPLoP	Control	53	68678	Other	4	2	3
EuroPLoP	Control	54	30924	Other	4	3	3.5
EuroPLoP	Control	54	41044	Other	4	3	3.5
EuroPLoP	Control	54	52975	Pattern	4	3	3.5
EuroPLoP	Control	54	53685	Pattern	5	2	3.5
EuroPLoP	Control	54	64601	Pattern	5	4	4.5
EuroPLoP	Control	58	13969	Other	2	1	1.5
EuroPLoP	Control	58	14458	Other	2	2	2
EuroPLoP	Control	58	18775	Other	3	1	2
EuroPLoP	Control	58	21221	Other	2	1	1.5
EuroPLoP	Control	58	26497	Other	1	2	1.5
EuroPLoP	Control	58	26994	Pattern	3	1	2
EuroPLoP	Control	58	34902	Pattern	2	1	1.5
EuroPLoP	Control	58	35617	Other	1	1	1
EuroPLoP	Control	58	44214	Other	1	1	1
EuroPLoP	Control	58	44467	Other	1	2	1.5
EuroPLoP	Control	58	47099	Other	2	2	2
EuroPLoP	Control	58	54821	Pattern	2	2	2
EuroPLoP	Control	58	73170	Other	2	1	1.5
EuroPLoP	Control	60	15787	Other	4	3	3.5
EuroPLoP	Control	60	25333	Other	2	2	2
EuroPLoP	Control	60	33726	Other	3	1	2
EuroPLoP	Control	60	33899	Other	4	1	2.5
EuroPLoP	Control	60	53821	Other	4	1	2.5
EuroPLoP	Pattern	55	21906	Pattern	3	2	2.5
EuroPLoP	Pattern	55	26522	Other	4	3	3.5
EuroPLoP	Pattern	55	26637	Other	4	2	3
EuroPLoP	Pattern	55	29399	Other	4	2	3
EuroPLoP	Pattern	55	35177	Other	4	2	3
EuroPLoP	Pattern	55	42037	Other	4	2	3
EuroPLoP	Pattern	55	58456	Other	3	2	2.5
EuroPLoP	Pattern	55	66133	Pattern	4	3	3.5
EuroPLoP	Pattern	55	77736	Other	5	2	3.5
EuroPLoP	Pattern	56	19520	Other	5	2	3.5
EuroPLoP	Pattern	56	29339	Other	4	1	2.5
EuroPLoP	Pattern	56	31499	Other	4	3	3.5
EuroPLoP	Pattern	56	44253	Pattern	4	3	3.5
EuroPLoP	Pattern	56	46178	Pattern	3	3	3
EuroPLoP	Pattern	56	68057	Pattern	3	2	2.5
EuroPLoP	Pattern	56	74785	Other	2	2	2
EuroPLoP	Pattern	57	14562	Pattern	1	2	1.5

Table A.1 – continued from previous page

Execution Grp.	Group	Part.	Dec.No	Type	Al. 1	Al. 2	Avg.
EuroPLoP	Pattern	57	21778	Pattern	4	1	2.5
EuroPLoP	Pattern	57	25077	Pattern	3	2	2.5
EuroPLoP	Pattern	57	31053	Pattern	3	2	2.5
EuroPLoP	Pattern	57	36371	Pattern	4	2	3
EuroPLoP	Pattern	57	47861	Pattern	4	2	3
EuroPLoP	Pattern	57	49896	Other	4	2	3
EuroPLoP	Pattern	57	58724	Pattern	4	2	3
EuroPLoP	Pattern	57	59375	Pattern	3	1	2
EuroPLoP	Pattern	57	64255	Pattern	2	2	2
EuroPLoP	Pattern	57	70273	Pattern	3	2	2.5
EuroPLoP	Pattern	57	71108	Pattern	4	1	2.5
EuroPLoP	Pattern	57	71172	Pattern	4	2	3
EuroPLoP	Pattern	57	72293	Other	3	2	2.5
EuroPLoP	Pattern	59	14001	Pattern	3	2	2.5
EuroPLoP	Pattern	59	16418	Pattern	3	2	2.5
EuroPLoP	Pattern	59	16548	Pattern	4	1	2.5
EuroPLoP	Pattern	59	18326	Pattern	4	3	3.5
EuroPLoP	Pattern	59	19641	Pattern	2	2	2
EuroPLoP	Pattern	59	27679	Pattern	3	2	2.5
EuroPLoP	Pattern	59	34381	Pattern	4	2	3
EuroPLoP	Pattern	59	36854	Pattern	3	2	2.5
EuroPLoP	Pattern	59	51205	Pattern	5	2	3.5
EuroPLoP	Pattern	59	70245	Pattern	3	1	2
EuroPLoP	Pattern	59	76602	Pattern	1	2	1.5
EuroPLoP	Pattern	61	22955	Pattern	3	1	2
EuroPLoP	Pattern	61	50589	Pattern	3	2	2.5
EuroPLoP	Pattern	61	50699	Pattern	3	2	2.5
EuroPLoP	Pattern	61	60055	Pattern	2	2	2
EuroPLoP	Pattern	61	62783	Pattern	3	2	2.5
EuroPLoP	Pattern	61	69068	Pattern	2	1	1.5
SWA Workshop	Control	P101	11958	Other	5	3	4
SWA Workshop	Control	P101	20924	Other	4	1	2.5
SWA Workshop	Control	P101	37389	Other	5	5	5
SWA Workshop	Control	P101	46596	Other	4	2	3
SWA Workshop	Control	P101	70401	Pattern	4	3	3.5
SWA Workshop	Control	P104	17130	Other	4	4	4
SWA Workshop	Control	P104	53026	Other	4	2	3
SWA Workshop	Control	P104	61606	Other	4	1	2.5
SWA Workshop	Control	P104	73623	Pattern	5	2	3.5
SWA Workshop	Control	P104	95115	Pattern	3	3	3
SWA Workshop	Control	P108	26170	Other	5	2	3.5
SWA Workshop	Control	P108	34797	Other	3	1	2

Table A.1 – continued from previous page

Execution Grp.	**Group**	**Part.**	**Dec.No**	**Type**	**Al. 1**	**Al. 2**	**Avg.**
SWA Workshop	Control	P108	48667	Other	4	2	3
SWA Workshop	Control	P108	53246	Other	4	2	3
SWA Workshop	Control	P108	80735	Other	5	5	5
SWA Workshop	Control	P108	88971	Pattern	3	3	3
SWA Workshop	Control	P109	23445	Other	2	2	2
SWA Workshop	Control	P109	33167	Other	3	1	2
SWA Workshop	Control	P109	38131	Other	2	1	1.5
SWA Workshop	Control	P109	48170	Pattern	2	2	2
SWA Workshop	Control	P109	79232	Pattern	1	1	1
SWA Workshop	Control	P109	81928	Other	2	1	1.5
SWA Workshop	Control	P109	87890	Other	3	3	3
SWA Workshop	Control	P109	93340	Other	1	1	1
SWA Workshop	Control	P112	34730	Pattern	2	2	2
SWA Workshop	Control	P112	57609	Pattern	3	1	2
SWA Workshop	Control	P112	59805	Other	2	2	2
SWA Workshop	Control	P112	68209	Other	4	2	3
SWA Workshop	Control	P112	76889	Pattern	3	1	2
SWA Workshop	Control	P112	97764	Pattern	2	1	1.5
SWA Workshop	Control	P113	38562	Other	1	1	1
SWA Workshop	Control	P113	51690	Pattern	1	1	1
SWA Workshop	Control	P113	53126	Other	1	1	1
SWA Workshop	Control	P113	70648	Other	1	1	1
SWA Workshop	Control	P113	81229	Pattern	1	1	1
SWA Workshop	Control	P116	33889	Other	1	1	1
SWA Workshop	Control	P116	53663	Other	1	1	1
SWA Workshop	Control	P116	64769	Other	1	1	1
SWA Workshop	Control	P116	71283	Pattern	1	1	1
SWA Workshop	Control	P116	80071	Other	1	1	1
SWA Workshop	Control	P117	66942	Pattern	1	1	1
SWA Workshop	Control	P117	81042	Other	3	2	2.5
SWA Workshop	Control	P120	32371	Other	4	2	3
SWA Workshop	Control	P120	49144	Other	2	1	1.5
SWA Workshop	Control	P125	25968	Other	5	3	4
SWA Workshop	Pattern	P102	14837	Pattern	4	2	3
SWA Workshop	Pattern	P102	47817	Pattern	2	1	1.5
SWA Workshop	Pattern	P102	75306	Other	2	2	2
SWA Workshop	Pattern	P102	92796	Other	3	1	2
SWA Workshop	Pattern	P103	21416	Pattern	4	3	3.5
SWA Workshop	Pattern	P103	26721	Pattern	4	3	3.5
SWA Workshop	Pattern	P103	53077	Pattern	3	2	2.5
SWA Workshop	Pattern	P103	79679	Other	3	2	2.5
SWA Workshop	Pattern	P105	34596	Pattern	3	1	2

Table A.1 – continued from previous page

Execution Grp.	Group	Part.	Dec.No	Type	Al. 1	Al. 2	Avg.
SWA Workshop	Pattern	P105	73625	Pattern	3	2	2.5
SWA Workshop	Pattern	P105	79704	Pattern	4	1	2.5
SWA Workshop	Pattern	P106	31527	Pattern	1	1	1
SWA Workshop	Pattern	P106	46146	Pattern	2	2	2
SWA Workshop	Pattern	P106	65149	Pattern	2	1	1.5
SWA Workshop	Pattern	P106	75358	Pattern	2	1	1.5
SWA Workshop	Pattern	P110	37008	Other	3	3	3
SWA Workshop	Pattern	P110	61530	Other	5	3	4
SWA Workshop	Pattern	P111	14680	Pattern	3	2	2.5
SWA Workshop	Pattern	P111	19077	Pattern	4	3	3.5
SWA Workshop	Pattern	P111	27818	Pattern	3	3	3
SWA Workshop	Pattern	P111	33776	Other	2	2	2
SWA Workshop	Pattern	P111	44350	Pattern	2	1	1.5
SWA Workshop	Pattern	P111	67646	Other	5	5	5
SWA Workshop	Pattern	P111	86855	Pattern	2	1	1.5
SWA Workshop	Pattern	P111	90696	Pattern	2	1	1.5
SWA Workshop	Pattern	P114	23961	Pattern	3	2	2.5
SWA Workshop	Pattern	P114	41389	Pattern	3	2	2.5
SWA Workshop	Pattern	P114	59052	Pattern	4	2	3
SWA Workshop	Pattern	P114	76798	Other	1	1	1
SWA Workshop	Pattern	P115	41047	Pattern	4	1	2.5
SWA Workshop	Pattern	P115	57170	Pattern	3	2	2.5
SWA Workshop	Pattern	P115	70149	Pattern	3	2	2.5
SWA Workshop	Pattern	P122	14967	Pattern	2	1	1.5
SWA Workshop	Pattern	P122	56783	Pattern	5	4	4.5
SWA Workshop	Pattern	P122	80909	Pattern	4	2	3
SWA Workshop	Pattern	P122	90464	Pattern	5	2	3.5
SWA Workshop	Pattern	P123	10498	Pattern	3	2	2.5
SWA Workshop	Pattern	P123	16680	Pattern	5	3	4
SWA Workshop	Pattern	P123	22025	Pattern	4	3	3.5
SWA Workshop	Pattern	P123	23757	Pattern	4	3	3.5
SWA Workshop	Pattern	P123	57463	Pattern	3	2	2.5
SWA Workshop	Pattern	P123	66453	Pattern	4	2	3
SWA Workshop	Pattern	P123	83213	Pattern	2	1	1.5

A.2 Typical decisions recovered by the participants

In the following, we present typical examples of decisions recovered by the participants. The first five decisions are pattern decisions, the latter five are of other types.

80909

Problem / Issue	Client calls instane method on server side
Decision	Use explicit invocation
Alternatives (optional)	Use implicit ~~xxx~~ invocation for async calls.
Arguments	The client can work on with the recieved result and use it in further calculations
Assumptions/ Constraints (optional)	Fast server response Low latency
Related requirements	Bidirectional (network) communication / Full Duplex
Related decisions (optional)	Explicit Invocation Pattern
Additional Comments (optional)	

Figure A.1: *Example 1 for pattern type decisions*

34730

Problem / Issue	Many different aspects need to be handled in one call.
Decision	Pipes & Filters (aspects)
Alternatives (optional)	
Arguments	Transparent hooking in of individual aspects
Assumptions/ Constraints (optional)	
Related requirements	
Related decisions (optional)	
Additional Comments (optional)	

Figure A.2: *Example 2 for pattern type decisions*

Your Participant Number **41417**

Problem / Issue	How should the EJB container create the implementations of EJBHome / EJBObject required by the Spec?
Decision	Use dynamic proxies to implement EJBHome and EJBObject.
Alternatives (optional)	Use a precompiler to generate the objects at deployment time.
Arguments	- code / proxy generation at runtime is less fragile than at deployment time - dynamic proxies are JDK standard mean
Assumptions/ Constraints (optional)	The construction of dynamic proxies via reflection is cheap compared to EJB deployment time
Related requirements	- improve system reliability - ensure minimized JBoss implementation effort by reusing standard JDK mean
Related decisions (optional)	
Additional Comments (optional)	

Figure A.3: *Example 3 for pattern type decisions*

Your Participant Number **66133**

Problem / Issue	how to manage complexity and divide application into group of subtasks?
Decision	Division to layers (instrumentation, agent, remote management)
Alternatives (optional)	
Arguments	Decomposition to parts not many boundaries to cross, no significant performance loss. Decomposition makes the system scalable => shorter time-to-market
Assumptions/ Constraints (optional)	
Related requirements	Maintainability - changeability (probably ~ trade-off with performance)
Related decisions (optional)	Layers - pattern
Additional Comments (optional)	

Figure A.4: *Example 4 for pattern type decisions*

Your Participant Number **51204**

Problem / Issue	The architect needs to decide on how to implement a micro kernel system
Decision	Use JMX as micro kernel architecture.
Alternatives (optional)	- implement custom micro kernel on your own - use some other framework, e.g. OSGI
Arguments	- JMX is a standardized Java technology, allowing for simple 3rd party integration - JMX exists already, lowering cost - JMX offers monitoring facilities
Assumptions/ Constraints (optional)	Architecture based on micro kernel
Related requirements	- Base development on standard technologies - Allow for runtime monitoring
Related decisions (optional)	Use micro kernel architecture. (#1) Use 100% Java (#5)
Additional Comments (optional)	

Figure A.5: *Example 5 for pattern type decisions*

Your Participant Number 68678

Problem / Issue	Which programming environment should be used for the JBoss implementation?
Decision	Implement JBoss on 100% pure Java
Alternatives (optional)	- integrate platform-specific code via JNI
Arguments	- pure JAVA runs on any JVM - only single platform code is compiled to - reduced testing effort for multiple platforms
Assumptions/ Constraints (optional)	- performance required can be done in 100% Java - all interfaces to required external resources (Network database) are accessible via Java API
Related requirements	- support multiple target platforms at minimal cost - reduce/unify required skill set of JBoss developers
Related decisions (optional)	Use JNX as micro kernel (#2)
Additional Comments (optional)	Might lead to performance problems; Testing multiple JVMs / Platforms is facilitated, but not unnecessary

Figure A.6: *Example 1 for other type decisions*

Your Participant Number 15787

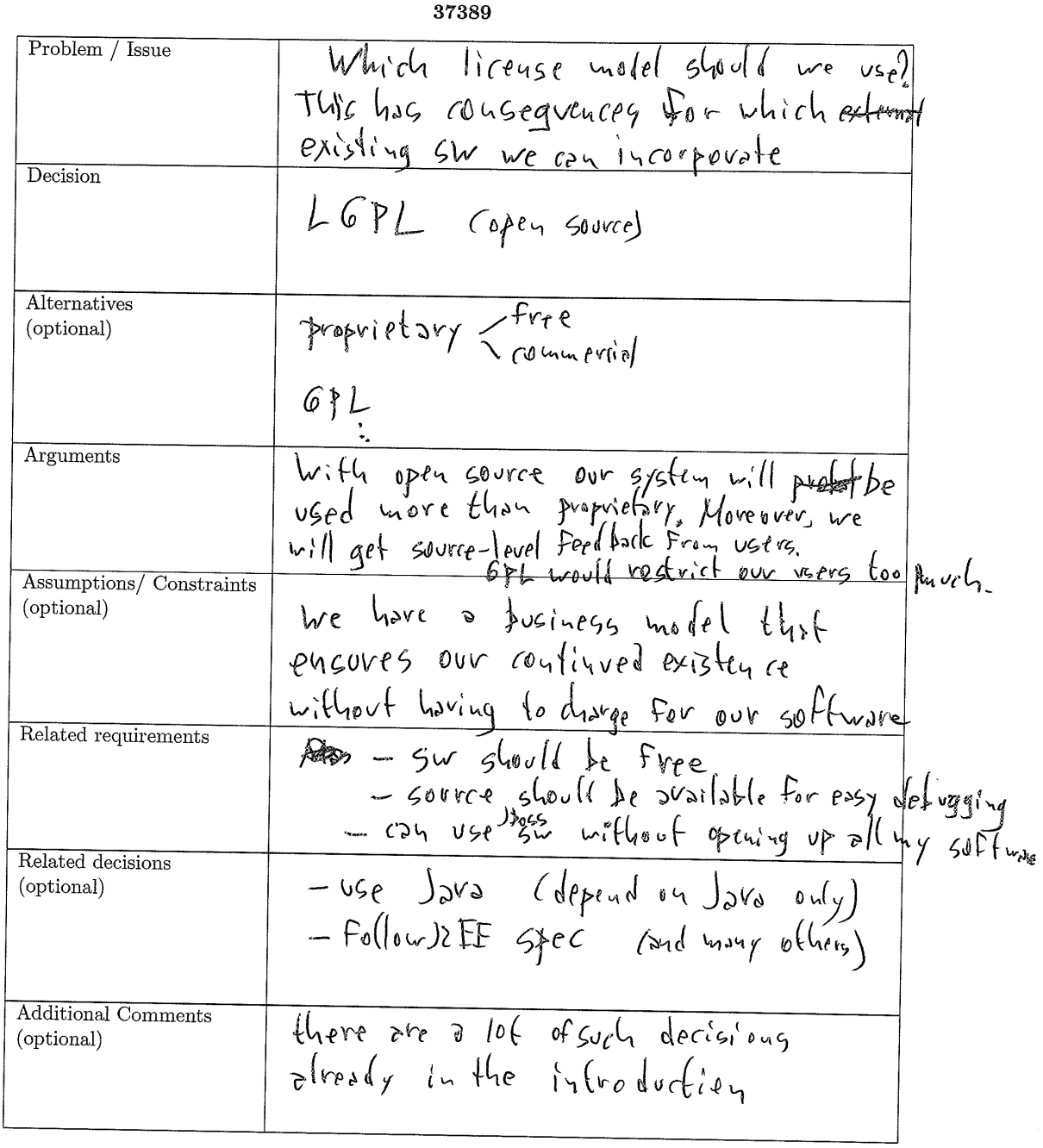
Problem / Issue	The Architect needed to provide access to MBean Server to (heterogeneous) remote Applications
Decision	Provide Access via Connectors & Adaptors
Alternatives (optional)	Define standard protocol & API for accessing the MBean Server
Arguments	Using connectors & Adapters Allows Any remote Application to access the MBean server, As long As that application supports one of communication frameworks (RMI...) & protocol (SNMP, ...). If A standard protocol is assumed, each application must be re-designed to support interaction with MBean server
Assumptions/ Constraints (optional)	For Each new communication framework / protocol A new connector / Adaptor must be provided.
Related requirements	• improves scalability / not sure about the term. I mean, you can Add other application that want to access the server. "Extensibility"? • Re-usability? (each time you Add new connector/Adaptor it can be re-used next time for other Applications using the same framework/protocol)
Related decisions (optional)	
Additional Comments (optional)	not All frameworks / protocols can be good for communication with the server.

Figure A.7: *Example 2 for other type decisions*

Your Participant Number **30924**

Problem / Issue	HOW TO PROVIDE ACCESS TO THE JBOSS INFRASTRUCTURE FOR REMOTE COMPONENTS, EACH USING A DIFFERENT REMOTING / PROTOCOL TECHNOLOGY ?. IN ADDITION, HOW CAN ONE KEEP THE NUMBER & KIND OF CONNECTORS EXTENSIBLE?
Decision	USING A KIND OF SERVICE ABSTRACTION ON TOP OF THE JBEAN SERVER
Alternatives (optional)	• STIPULATE A SINGLE REMOTING STRATGY / CHANNEL, REMOTE ENDS HAVE TO COMPLY WITH •
Arguments	• INCREASE REACH & INTEGRATABILITY IN HETEROGENEOUS SYSTEM LANDSCAPES → e.g. in in corporate context • REDUCE POTENTIAL ENTRY / ADAPTATION COSTS PER CUSTOMER → AVOID ON-SITE INTEGRATION EFFORTS • REDUCE DEVELOPMENT EFFORT TO FREESTANDING EXTENSIONS IF A NEW TECHNOLOGY ENERGY IS REQUIRED
Assumptions/ Constraints (optional)	• TARGETED SYSTEM LANDSCAPE DO IS BASED ON ESTABLISHED REMOTING STANDARDS (NO AD-HOC OR PROPRIETARY SOLUTIONS)
Related requirements	• INCREASED INTEGRATABILITY (LEGACY INTEGRATION) • COPING WITH SYSTEM HETEROGENEITY • AVOID LOCK-INS ; EASEN FURTHER-DEVELOPMENT OF THE OVERALL INFRASTRUCTURE (i.e., allow for TECHNOLOGY CHANGE IN REMOTING)
Related decisions (optional)	• SEE PAGE (2) : "LAYERING OF MESSAGE INFRASTRUCTURE" (JMX)
Additional Comments (optional)	

Figure A.8: *Example 3 for other type decisions*



37389

Problem / Issue	Which license model should we use? This has consequences for which existing SW we can incorporate
Decision	LGPL (open source)
Alternatives (optional)	proprietary / free \ commercial GPL ⋮
Arguments	With open source our system will be used more than proprietary. Moreover, we will get source-level feedback from users. GPL would restrict our users too much.
Assumptions/ Constraints (optional)	We have a business model that ensures our continued existence without having to charge for our software
Related requirements	– SW should be free – source should be available for easy debugging – can use Jboss SW without opening up all my software
Related decisions (optional)	– use Java (depend on Java only) – Follow J2EE spec (and many others)
Additional Comments (optional)	there are a lot of such decisions already in the introduction

Figure A.9: *Example 4 for other type decisions*

11958

Problem / Issue	How to keep system development and understanding manageable?
Decision	Subdivide into modules (EJB container, JBossTx, ...)
Alternatives (optional)	- use a thick soup of classes - use a different module subdivision
Arguments	A subdivision into modules helps to distribute development effort and allows incremental learning of the modules.
Assumptions/ Constraints (optional)	
Related requirements	- system should be easy to learn For users - development effort should stay within reasonable limits
Related decisions (optional)	
Additional Comments (optional)	

Figure A.10: *Example 5 for other type decisions*

Appendix B

Appendix to Chapter 6

B.1 Concern analysis

Table B.1 shows the outcome of the concern analysis described in Section 6.2. Each row contains an architectural knowledge management (AKM) use case elicited from the literature (Liang et al. 2009, Kruchten et al. 2006, Jansen et al. 2007), the concerns derived from these use cases (please refer to Table 6.1 for a description of the concerns), and typical stakeholders having those concerns. Additionally, the table indicates how the concerns were derived (column *DER*). The table is ordered by the publications, from which the use cases were elicited.

The following activities were used to derive concerns:

- **Derive (DER)**: A concern, or a set of concerns was derived from a decision-related use case.
- **Project (PRJ)**: A use case that does not directly involve architecture decisions was projected to architecture decision concerns.
- **Complement (COM)**: A new concern was introduced to complement concerns derived from a use case.

The analysis was done by the three authors. In cases, where the three authors identified different concerns, a discussion took place to reach consensus.

Table B.1: Decision concerns derived from use cases

Use Case	Derived concerns	DER	Typical stakeholders
If we want to do a change in an element, what are the elements impacted (decisions, and elements of design). (Kruchten et al. 2006)	C13	PRJ	Architects
Find out if multiple systems can be combined (migrated) (Kruchten et al. 2006)	C15, C14, C9, C8	PRJ, COM	Architects, Reviewers
From a given perspective (such as security, safety, reuse, etc.) what are the knowledge elements involved? (Kruchten et al. 2006)	C6	PRJ	Architects, Reviewers, Customers, Requirements Engineers
You want to integrate multiple systems and decide whether they fit. The tool would help answering questions about integration strategies. (Kruchten et al. 2006)	C15, C14, C9, C8	PRJ, COM	Architects, Reviewers
What pieces of Architectural Knowledge have been added or modified since the last review? (Kruchten et al. 2006)	C21, C22	PRJ, COM	Architects, Reviewers

Table B.1 – continued from previous page

Use Case	Derived concerns	Derivation Activity	Typical stakeholders
The architect makes sure that all the dependencies of removed AK (i.e., the consequences of an architecture decision) have been removed as well. (Kruchten et al. 2006)	C11, C22, C8	PRJ, COM	Architects, Reviewers
What pieces of Architectural Knowledge have been added or modified since the last review? (Kruchten et al. 2006)	C21	PRJ	Architects, Reviewers
Over a time line, find what the sequence of design decisions has been. (Kruchten et al. 2006)	C20	DER	Architects, Reviewers, New Project Members
Identify decisions being hubs (god decisions). (Kruchten et al. 2006)	C10, C12	DER, COM	Architects, Reviewers
Identify circular dependencies. (Kruchten et al. 2006)	C10	DER	Architects, Reviewers
Identify decisions that gain weight over time and are more difficult to change or remove. (Kruchten et al. 2006)	C10, C21	DER, COM	Architects, Reviewers
Identify the stakeholder who seems to have the most "weight" on the decisions, and who therefore maybe the one that could be most affected by the future evolution of the system. (Kruchten et al. 2006)	C16, C18, C19, C17	DER, COM	Architects, Reviewers, Manager
Identify who are the stakeholders whose changes of mind are doing the most damage to the system. (Kruchten et al. 2006)	C18, C21, C22	PRJ, COM	Architects, Reviewers, Managers
Identify patterns in the decision graphs that can be a useful fashion and lead to guidelines for the architects. (Kruchten et al. 2006)	C23, C5	DER, COM	Architects, Reviewers, Domain Experts
Trace between various AK elements, e.g. design decisions, rationale, and design. (Liang and Avgeriou 2009)	C11, C6, C4, C3, C19, C5, C17	PRJ, COM	Architects, Reviewers, Customers, Requirements Engineers, New Project Members
The reviewer performs a critical evaluation of the AK, e.g. to make sure that requirements have been satisfied in the architecture design. (Liang and Avgeriou 2009)	C6, C7, C4, C3, C2	PRJ, COM	Architects Reviewers, Customer, New Project Members
Perform an evaluation of architectural knowledge. (Liang and Avgeriou 2009)	C4, C3, C5	PRJ, COM	Architects, Reviewers, Customers, New Project Members

Table B.1 – continued from previous page

Use Case	Derived concerns	Derivation Activity	Typical stakeholders
The architect evaluates when the architecture can be considered as finished, complete, and consistent, e.g. verify whether a system conforming to the architecture can be made or bought. (Liang and Avgeriou 2009)	C6, C7, C9, C8, C2	PRJ, COM	Architects, Reviewers
Browse architectural knowledge dependencies. (Liang and Avgeriou 2009)	C10	PRJ	Architects, Reviewers
Browse architectural knowledge traces. (Liang and Avgeriou 2009)	C11, C16, C12, C6, C4, C8, C19, C5, C17	PRJ, COM	Architects, Reviewers, Customers, Managers
Understand the rationale of a design decision. (Liang and Avgeriou 2009)	C3, C5	DER, COM	Architects, Reviewers, Customers, New Project Members
Distill specific knowledge from a system into general knowledge (e.g. architecture pattern) that can be reused in future systems. (Liang and Avgeriou 2009)	C23, C5	DER, COM	Architects, Reviewers, Domain Experts
Produce a consistent subset of Architectural Knowledge to prime the pump for a new system (reuse Architectural Knowledge). (Liang and Avgeriou 2009)	C23, C5	DER, COM	Architects, Reviewers, Domain Experts

B.2 Decision views from the case study

Figures B.1, B.2, and B.3 contain details of decision views created during the case study.

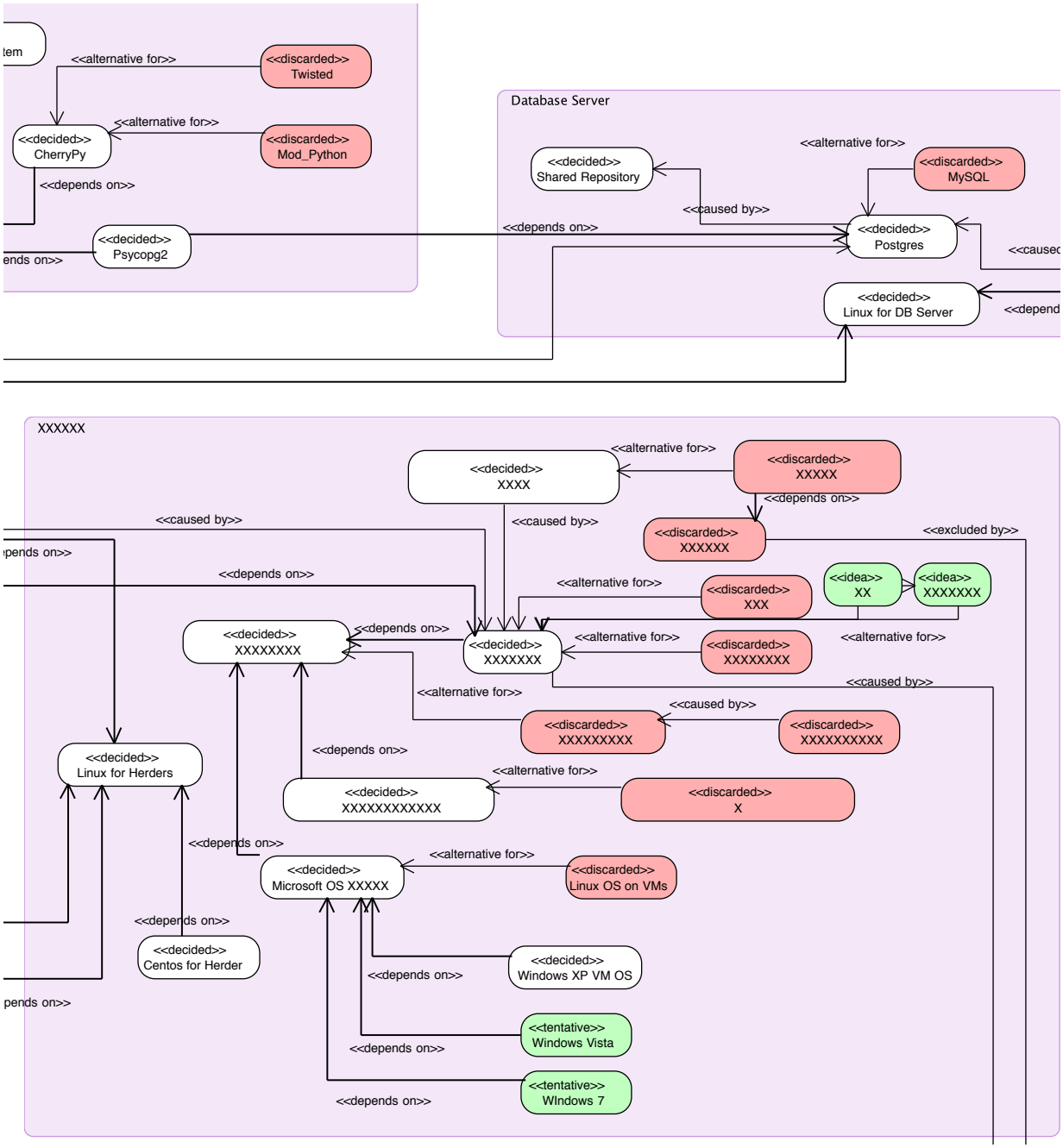


Figure B.1: *Partially censored excerpt from a relationship view*

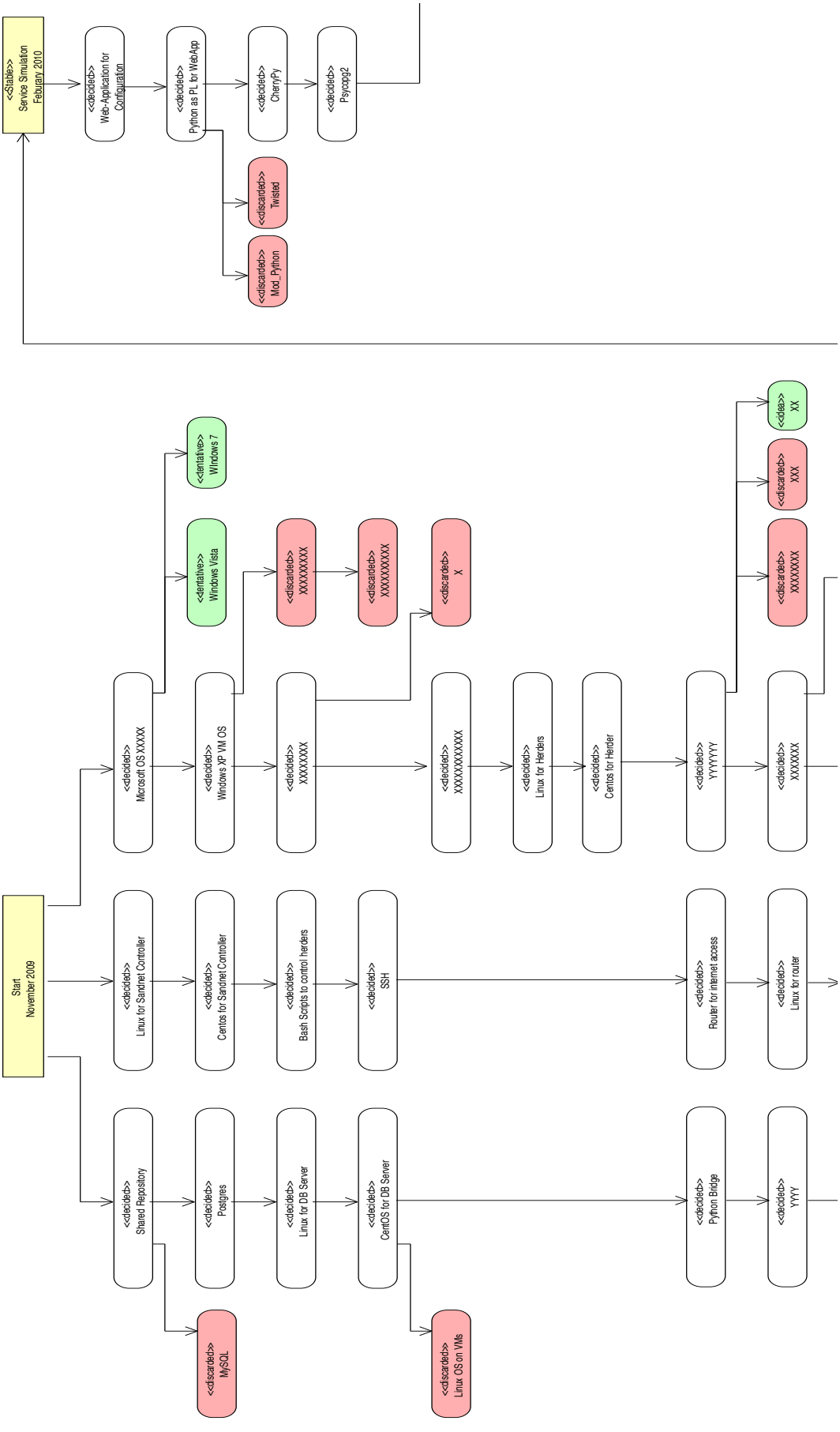


Figure B.2: *Partially censored excerpt from a chronology view*

<table>
<tr><td>Name</td><td>Postgres</td></tr>
<tr><td>Current Version</td><td>1 (Service Simulation <<Release>>)</td></tr>
<tr><td>Current State</td><td>Decided</td></tr>
<tr><td>Decision Group</td><td>Database Server</td></tr>
<tr><td>Problem/Issue</td><td>A DBMS has to be chosen to implement the shared repository.</td></tr>
<tr><td>Decision</td><td>Use PostgreSQL

http://www.postgresql.org</td></tr>
<tr><td>Alternatives</td><td><MySQL></td></tr>
<tr><td>Arguments</td><td>IP4R datatype for storing IP adresses, good experience from prior projects regarding performance, expertise in administration and configuration present, extendable (PL functions, new data types etc.)</td></tr>
<tr><td>Related decisions</td><td>• Psycopg2 <<depends on>> This
• This <<caused by>> Shared Repository
• CentOS <<caused by>> This
• Bash scripts to control herders <<caused by>> This</td></tr>
<tr><td>Related requirements</td><td></td></tr>
<tr><td>History</td><td>
<table>
<tr><th>Stakeholder</th><th>Action</th><th>Status</th><th>Iteration</th></tr>
<tr><td>XXX <<Architect>></td><td><<Propose>></td><td><<Tentative>></td><td>Service Simulation</td></tr>
<tr><td>XXX <<Architect>></td><td><<Validate>></td><td><<Decided>></td><td>Service Simulation</td></tr>
</table>
</td></tr>
</table>

Figure B.3: *A single decision from the case study in the detail view*

B.3 Viewpoint definitions and correspondence rules

B.3.1 Decision framework metamodel

Figure B.4 shows a shared metamodel for the decision viewpoint elements. The metamodel is not specific to one particular viewpoint; instead, it is common to all decision viewpoints introduced in Chapter 6. Elements with a gray background map to the corresponding elements in Figures 2 and 4 of ISO/IEC/IEEE 42010 (ISO/IEC/IEEE 2011). Therefore, the architecture description elements used in the architecture decision viewpoints (white background) integrate seamlessly into the conceptual framework of the standard.

A shared metamodel, together with well-defined constraints and correspondence rules, can ensure consistency among the views from different viewpoints. The intra-model constraints will be defined later, as a part of the viewpoint definitions. Additionally, inter-model and inter-view correspondence rules are defined to ensure consistency between the views.

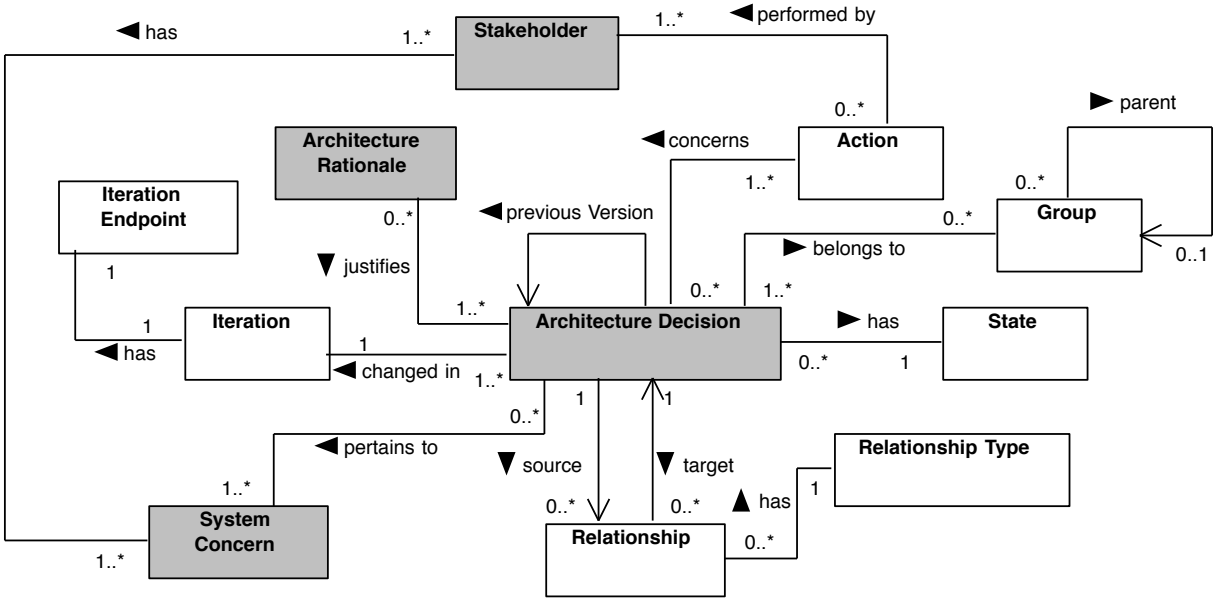


Figure B.4: *Metamodel of decision viewpoints*

B.3.2 Decision relationship viewpoint

As mentioned in Section 6.3, the relationship viewpoint describes relationships between architectural design decisions. Table 6.3 shows the concerns framed by the viewpoint, as related to the mentioned stakeholders.

Model kind

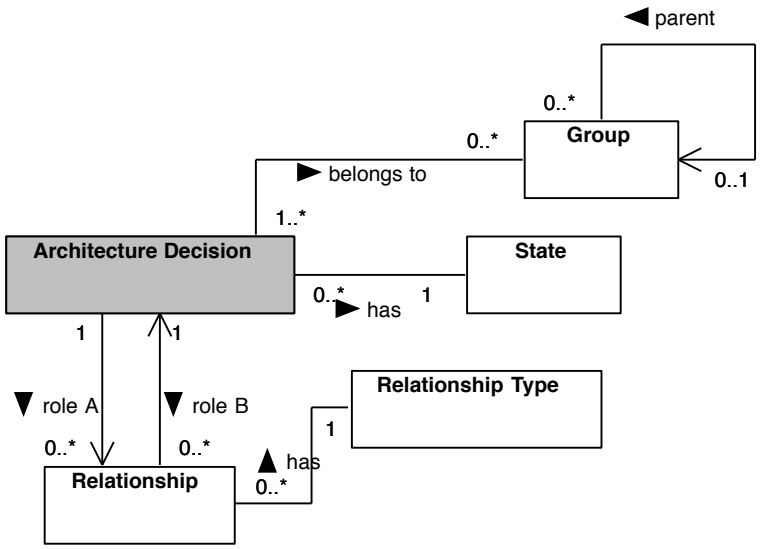


Figure B.5: *Metamodel of decision relationship viewpoint*

Figure B.5 shows a metamodel for the relationship viewpoint. It documents the model kind, which presents the conceptual elements for architecture models that ad-

here to it. It uses the notation for class diagrams from the Unified Modeling Language (UML). One relationship view can contain multiple relationship models of this model kind; however, every decision is represented only once in a view.

An architecture decision is identified by a short name. Although an architecture decision has potentially many versions, one for every state change, the relationship view contains only the current versions of the decisions shown. A decision has a state, which

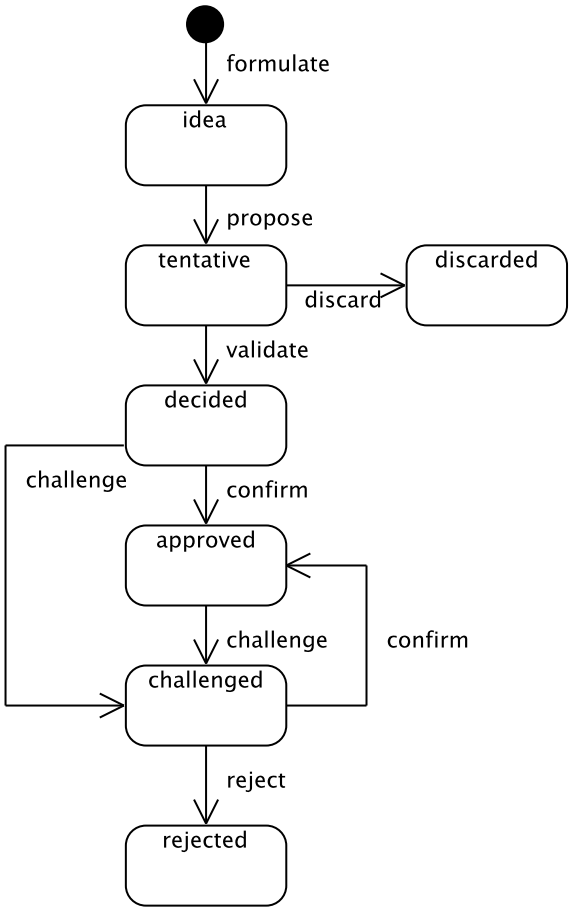


Figure B.6: *UML diagram for state changes of architecture decisions*

can be freely chosen depending on the needs of the respective development project. All possible states must be clearly specified prior to being used. With one exception, we adopt the decision states from Kruchten's ontology of architectural design decisions (Kruchten 2004a):

- **Idea**: This state is used for decisions which are just loose ideas that architects want to document so that they don't get lost. If a decision has the idea-state, then it cannot have any relationships to other decisions.
- **Tentative**: This state is used for decisions that are seriously considered by the architect.
- **Decided**: The decision reflects the current position of the architect and must be consistent with other "decided decisions".

- **Approved**: This state is reached, if a previously decided decision has been confirmed; for instance during a review or a customer meeting.

- **Challenged**: This state is applicable, if a stakeholder raises issues about a previously decided or approved decision.

- **Rejected**: A rejected decision is a decision that was challenged and has been removed from the current iteration of the architecture. For the sake of simplicity, we subsume Kruchten's **Obsolesced**-state under this state as well.

In addition to Kruchten's states, we define the state *discarded*. A discarded decision is a formerly tentative decision that was not decided, for instance a design option that was not chosen among the considered alternatives. Figure B.6 shows the decision states along with the respective state transitions.

Decisions participate in relationships. Every relationship refers to exactly two decisions, one source and one target decision. For instance, decision1 (source) replaces decision2 (target). For the sake of simplicity, the meta model only takes binary relationships into account, although in some cases n-ary relationships between decisions may be useful.

A relationship has a specific relationship type, which again can be freely chosen, but should be clearly specified. We define the following relationship types:

- **Depends on**: If decision B depends on decision A, then B cannot be decided or approved without A being in that state. Expressed the other way round, A is a prerequisite for B. This relationship includes Kruchten's cases in which decision B is part of a decomposition of A, or if B is comprised by A.

- **Caused by**: If decision B is caused by decision A, then B would not have been decided without A being decided. This relationship expresses causality, without imposing further constraints on the decisions.

- **Is excluded by**: Decision A is excluded by decision B, if A cannot be decided as long as decision B is decided. In other words, decision B prevents decision A.

- **Replaces**: Decision B replaces decision A, if B was put into practice instead of A.

- **Is alternative for**: If decision B is an alternative for decision A, then B was considered as design option instead of A. Two decisions are alternatives when they address a significant common set of requirements.

Table B.2 shows a mapping of our types to the relationship types defined in Kruchten's ontology (Kruchten 2004a). An arrow after the relationship type name indicates that the relationship types in that row are complementary—they may be expressed in either of two ways: e.g., if decision A was *caused by* decision B, then decision B *enables* decision A.

Table B.2: *Mapping of relationship types to Kruchten's ontology*

Used type	**Kruchten's type**
Caused by	No match
Depends on	Enables ←
	Decomposes
	Subsumes ←
	Comprises ←
Replaces	No match
Is alternative for	Is an alternative to
Is excluded by	Forbids ←
No match	Conflicts with
	Constrains ←
	Overrides

A decision can belong to zero or more decision groups. This allows for logical grouping of decisions according to self-defined characteristics. For instance, decisions could be grouped by subsystem, use-case package, physical location, or component. Decision groups can have parent groups. This is especially helpful to organize the documentation of large numbers of decisions. The models in the relationship view can provide different "scales", e.g., one model showing only the root decision groups and their relationships, and additional models for "zooming into" each of the groups showing either decisions or subgroups, which themselves contain decisions or further subgroups.

The following constraints apply to the elements within this model kind:

1. The architecture decisions shown in one relationship view all refer to the same point in time.
2. Every decision occurs exactly once.
3. A decision has a unique name and exactly one state.
4. A decision can participate in zero or more relationships.
5. A relationship has exactly one type.
6. A relationship has exactly two non-identical endpoints.
7. A relationship model showing decision groups without associated decisions must be refined by one or more additional relationship models showing which decisions belong to which decision group.
8. *Caused by*-relationships cannot point to *idea* or *discarded* decisions.
9. *Caused by*-relationships cannot originate from *idea* decisions.

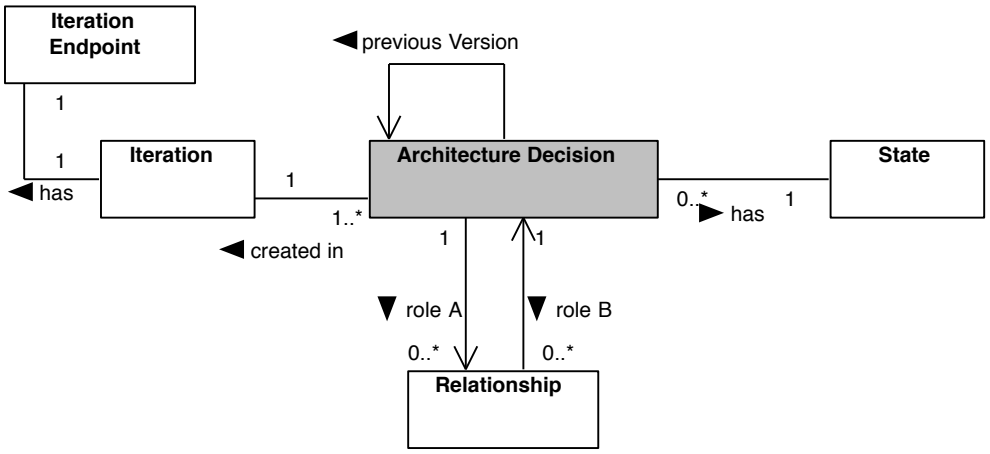


Figure B.7: *Metamodel of chronology viewpoint*

10. *Depends on*-relationships can only point to *tentative*, *decided*, *approved* or *challenged* decisions.

11. *Depends on*-relationships cannot originate from *idea* decisions.

12. *Excluded by*-relationships cannot point to *idea*, *tentative*, *discarded* or *rejected* decisions.

13. *Excluded by*-relationships can only originate from *idea*, *tentative*, *discarded* or *rejected* decisions.

14. *Replaces*-relationships can only point to *rejected* decisions.

15. *Replaces*-relationships cannot originate from *idea* decisions.

16. *Alternative for*-relationships cannot point to *idea* or *discarded* decisions.

17. *Alternative for*-relationships can only originate from *tentative* or *discarded* decisions.

Note that the presented constraints refer to the used decision states and relationship types. If different states or relationship types are used, then the constraints must be revised accordingly. In addition to the internal model constraints presented above, cross-viewpoint correspondence rules exist. These rules will be presented in B.3.6.

B.3.3 Decision chronology viewpoint

This viewpoint shows the evolution of architecture decisions in chronological order. Table 6.5 shows the concerns framed by this viewpoint related to the respective stakeholders.

Model kind

Figure B.7 shows a metamodel for the chronology viewpoint. It documents the model kind, which presents the conceptual elements for architecture models that adhere to it. Again, the notation for UML class diagrams is used. An architecture decision is made or changed (i.e., a state change) within an architecture iteration. We define iterations as versions of the architecture as a whole. An iteration endpoint has a date and furthermore a type that can be freely chosen. We propose the following three predefined types:

Milestone: A version of the architecture that has reached a stable state (or an intermediate stable state).

Release: A version of the architecture that is delivered to a customer or made available to the public for use.

Snapshot: A snapshot can be incomplete and possibly inconsistent. This iteration endpoint can be used to express that a customer or project team meeting took place where some decisions were made or discussed without ending up with a stable iteration version.

The following constraints apply to the elements within this model kind:

1. Every decision has a unique name and exactly one state at a time.
2. Every decision can take role A in zero or more relationships (role B is followed by role A).
3. Every decision can take role B in zero or more relationships (role B is followed by role A).
4. Every relationship has the type *followed by*.
5. Every relationship has exactly two non-identical endpoints.
6. Decision states can only change in conformance to the state diagram shown in Figure B.6.
7. Every iteration has exactly one endpoint with a unique name (e.g. Iteration 4).
8. Concurrent decision paths (e.g. by different architects making decisions autonomously in the same project) cannot cross the boundaries of iterations (marked by an iteration endpoint).

In addition to the internal model constraints presented above, cross-viewpoint correspondence rules exist. These rules will be presented in B.3.6. An extract of a model that corresponds to this model kind is shown in Figure 6.4.

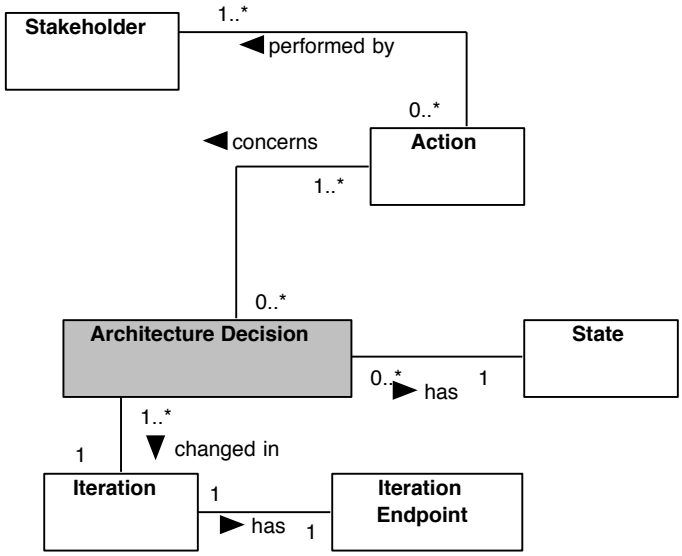


Figure B.8: *Metamodel of stakeholder involvement viewpoint*

B.3.4 Decision stakeholder involvement viewpoint

The stakeholder involvement viewpoint shows the responsibilities of relevant stakeholders in the decision-making process. Table 6.4 shows the concerns framed by this viewpoint related to the respective stakeholders.

Model kind

Figure B.8 shows a meta model for the stakeholder involvement viewpoint. It documents the model kind, which presents the conceptual elements for architecture models that adhere to it. Corresponding elements of the chronological metamodel have the same semantics as in this viewpoint. Every architecture decision is caused by at least one action performed by a stakeholder. In larger projects, the stakeholder can represent a group, or organization, e.g. a development team or a department in a company. The actions can be adapted to the needs of a concrete software project. In our examples we use the following actions:

formulate: A decision is documented as a rough idea that should be revisited in the future. The corresponding decision state is *idea*.

propose: A new decision or a set of new decisions is proposed by an architect. The corresponding decision state is *tentative*.

discard: A tentative decision is discarded by an architect. The corresponding decision state is *discarded*. A discarded decision has never reached a state higher than tentative.

validate: A decision, or a set of decisions, was validated by a stakeholder. The corresponding decision state is *decided*.

confirm: A decision, or a set of decisions was confirmed by a stakeholder on the customer site. The corresponding decision state is *approved*. This action can be performed on a challenged decision to (re-)confirm it, or on a decided decision.

challenge: A decision or a set of decisions is challenged by a stakeholder. The corresponding decision state is *challenged*.

reject: A decision that was challenged before is rejected. The corresponding decision state is *rejected*.

Figure B.6 shows stakeholder actions and corresponding decision state transitions.

A stakeholder can have one or more roles in a project. The roles depend on the circumstances in a concrete project. They must be clearly defined prior to being used. We used the following stakeholder roles:

architect: A person or organizational unit responsible for making architecturally relevant decisions in a project.

manager: A person or organizational unit who is responsible for the project in a company.

customer: A person or organizational unit serving as customer representative who is in charge of confirming architecture decisions.

The following constraints apply to the elements of this model kind:

1. Every decision has a unique name.
2. Every iteration endpoint has a unique name.
3. All decision versions changed in one iteration are shown.
4. Every stakeholder has a unique name and zero or more stakeholder roles.
5. Every stakeholder shown performed at least one action.
6. Every action has exactly two non-identical endpoints.
7. Every action originates from exactly one stakeholder in a role.
8. Every action points to a decision, or an iteration endpoint. If the target is an iteration endpoint, then the corresponding action is performed for all decisions (respectively decision versions) changed in that iteration.

In addition to the internal model constraints presented above, cross-viewpoint correspondence rules exist. These rules will be presented in B.3.6. An example of a model that corresponds to this model kind is shown in Figure 6.3.

B.3.5 Decision detail viewpoint

The decision detail viewpoint provides an in-depth textual description of each architecture decision documented in a software project. Table 6.6 shows the concerns framed by this viewpoint related to the respective stakeholders.

Model kind

The metamodel for the decision details viewpoint is identical to the shared metamodel for all viewpoints shown in Figure B.4. In addition to the elements that were already described in the other viewpoint definitions, the model contains a relationship between architecture decision and system concerns. Every architecture decision is represented by exactly one decision detail model. Ideally, the total of decision detail models shows every architecture decision documented for a system. An example of a model that corresponds to this model kind is shown in Figure 6.5.

B.3.6 Correspondences between viewpoints

The documentation framework for architecture decisions is comprised of four viewpoints. A view conforming to one of these viewpoints is composed of one or more models. The fact that the same subject is represented in multiple independent models creates the risk of inconsistencies. The new ISO/IEC/IEEE 42010 standard for architecture documentation introduces *correspondences* to express cross-model relationships between architecture description elements (ISO/IEC/IEEE 2011). In the following, we define a number of correspondence rules, which have to be observed by views of the respective viewpoints in order to be consistent. In combination with the correspondence rules, we use a shared metamodel for all model kinds to ensure cross-model consistency. The shared metamodel was introduced in B.3.1. The correspondence rules are expressed in terms of constraints and relationships of the architecture models and description elements defined in the metamodel. Note that some of the rules are only applicable if the framework is used as a whole. If the framework or individual viewpoints are customized, then the rules must be revised accordingly.

With the exception of the chronology view, all views are potentially comprised of more than one model. A chronology view comprises one model showing the evolution of all decisions made in the system to document. Please refer to the respective viewpoint sections for more information about internal viewpoint constraints.

R1: The total number of relationship models contains all latest versions of every architecture decisions made in a system. The latest versions must correspond to the latest occurrence of a decision in the chronological model.

R2: A stakeholder involvement model must exist for every iteration shown in the chronological model. Every stakeholder involvement model must contain the versions of architecture decisions belonging to the respective iteration.

R3: A decision detail model contains all incoming and outgoing relationships of a decision shown in the relationship models.

R4: The current state of a decision in the decision detail model must correspond to the state of the latest occurrence of the decision in the chronological model.

R5: The alternatives mentioned in one decision detail model must be identical to the decisions in the relationship view having an *is alternative for* relationship pointing to the decision represented in the model.

R6: The history of a decision represented in the decision detail model must contain all stakeholder actions performed on that decision shown in all stakeholder involvement models.

B.4 Example of qualitative analysis process

In Section 6.4, we described the procedure used to qualitatively analyze parts of the data gathered in the case study. In the following, an example of the analysis process is given. It is taken from the transcript of the focus group conducted after the architecture review. It was chosen, because it reflects the typical procedure we used for the qualitative analysis:

Original comment given by one of the domain experts:"I liked the relationship view. I could make use of it quite well. Especially the relationships and what would happen if I changed something. I think this is more clearly illustrated than in any table. This is great progress and I was clearly impressed."

This passage was labelled with *Research question two* and *relationship view*. It is noticeable that the comment is hard to interpret when taken out of the context. The commenter is referring to the relationship view of the sandnet project, which he was showing to the other participants while talking. He mentions the different relationships between decisions and emphasizes that the relationships can be used to analyze which decisions would be impacted if a specific decision changed. Then he compares the relationship view with a "table". Here, he refers to a decision table, which strictly speaking is a model in a decision detail view. From this comment, we derived the following statements:

- Relationship views illustrate the relationships between decisions.
- Relationship views support impact analysis.
- Relationship views illustrate decision relationships better than decision detail views.

B.5 Question guide used during the focus group

The following set of questions was used as orientation by the moderator of the focus group, which took place after the review. The questions were not necessarily asked by the moderator, nor were they answered in a specific order. During an open discussion between the participants, the moderator made sure that the participants gave enough information so that the questions could be answered. The focus group data collection method was described in Section 6.4.2.

- How did the views support the participants in understanding the architecture?
  - Which information were they missing?
  - Which information did they get?
- How did the views help them to communicate architecture, what was missing?
- How do they usually document architecture?
  - What are the liabilities and benefits of the decision views compared to their usual way of doing it?
- Which concerns do they have in architecture documentation in general?
  - How did the relationship view support them, what was missing?
  - How did the chronology view support them, what was missing?
  - How did the documented decisions support them, what was missing?
- Which concerns do the participants have in architecture documentation when starting a new project?
  - How did the relationship view support them, what was missing?
  - How did the chronology view support them, what was missing?
  - How did the documented decisions support them, what was missing?
- Which concerns do they have in architecture documentation when doing architecture reviews? For identifying decisions/ sensitivity points/ trade-off points and risks?
  - How did the relationship view support them, what was missing?
  - How did the chronology view support them, what was missing?
  - How did the documented decisions support them, what was missing?
- Which concerns do they have in architecture documentation during architecture evolution?
  - How did the relationship view support them, what was missing?

- How did the chronology view support them, what was missing?
- How did the documented decisions support them, what was missing?

Appendix C

Appendix to Chapter 7

C.1 Integration of the forces viewpoint into the decision framework's metamodel

In order to integrate the forces viewpoint, presented in Chapter 7, into the previously developed framework for architecture decisions, defined in Chapter B.3, a few changes to the shared metamodel had to be made. Figure C.1 shows the adapted metamodel. Changed classes were marked with a colored background.

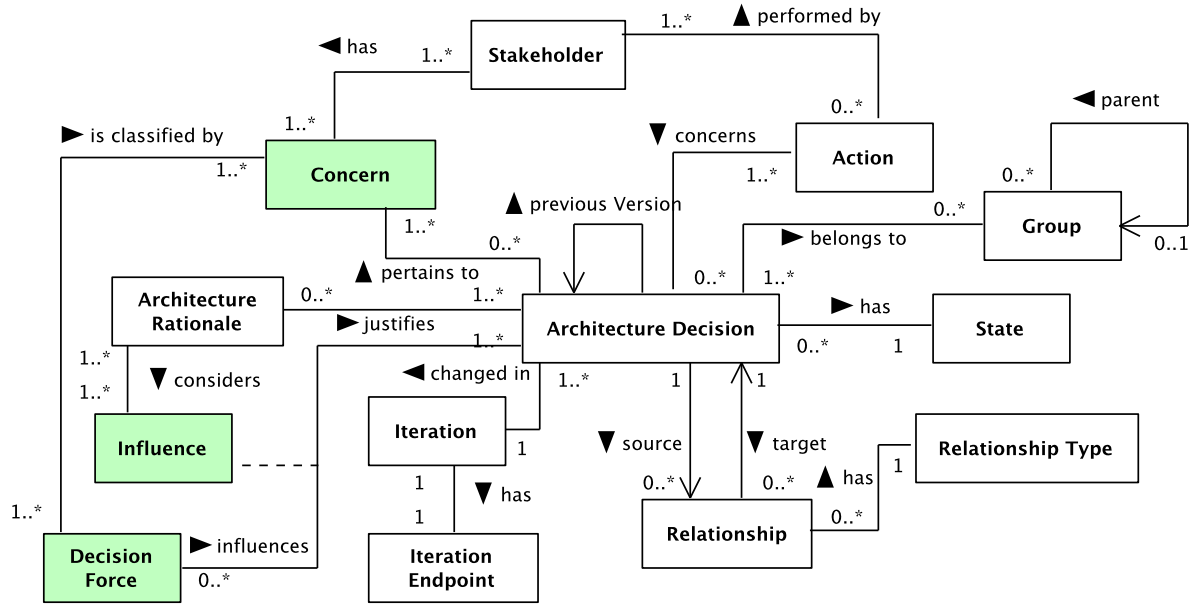


Figure C.1: *Adapted metamodel of the decision framework after the integration of the forces viewpoint.*

The class *System Concern*, in the original metamodel, was renamed to *Concern*, following the final revision of ISO/IEC/IEEE 42010 (ISO/IEC/IEEE 2011), which the framework is based on. Furthermore, two additional classes were added and linked with classes from the original metamodel. The class *Decision Force* represents a single force. It has an influence on one or more architecture decisions. The influence relationship is qualified by the *Influence* class. The influence of the forces on an architecture decision should be considered in the decision's *Architecture Rationale*. As further described in Chapter 7, each force is classified by at least one concern.

The described changes to the metamodel are backwards compatible. Existing views of the other viewpoints are not affected by these changes.

C.2 Constraints for the forces viewpoint's model kind

The following constraints apply to the elements within the force's viewpoint's model kind:

1. Within one decision topic, there can only be one decision with a state *decided*, or

above.

2. The architecture decisions shown in one forces view all refer to the same point in time.

3. A decision has a unique name and exactly one state.

4. A force has a unique code and a description.

5. A concern has a unique name.

In addition to the internal model constraints presented above, additional correspondence rules exist for the integration with the other viewpoints (so called cross-viewpoint correspondence rules). These rules are presented in the next section.

C.3 Cross-viewpoint correspondence rules

The following correspondence rules are a supplement to the correspondence rules defined in Section B.3.6. The numbering scheme from this section was adopted, therefore the following rules start with number seven:

R7: All concerns mentioned in decision detail models must exist in the decision forces models.

R8: The decision states shown in the decision forces models must correspond to the latest states of the respective decision in the chronological view.

R9: All decisions within a decision topic in a forces model that have a lower state than *decided*, must have *alternative-for* relationships with the one decision in the decision topic that has a state equal to, or higher than *decided*.

Appendix D

Appendix to Chapter 8

D.1 Question guide used during the weekly focus groups

The following questions were used as orientation for the moderator of the focus group to make sure that the generally open discussions cover all important aspects of interest. The questions were neither asked verbatim or directly, nor were they necessarily covered in a specific order.

- What has the team done since the last focus group?
- How did they elicit requirements?
- What are the main requirements?
- How do they document requirements?
- Did the team negotiate requirements with the customer?
- How do they prioritize requirements, and which requirements were regarded first and for which reasons?
- Which decisions have been made, and which alternatives were considered?
- How do they make decisions?
- Do the team members challenge each other a lot?
- How does the team lead design discussions?
- Which media, apart from the whiteboard, are used during design discussions?
- What is the team's confidence in the soundness of the decisions? Where are uncertainties?
- Did the team make any assumptions? Which assumptions and why?
- Does the team try to avoid complexity? How?
- Did they make trade-offs between multiple requirements?
- Did they create prototypes, and if so what were they used for?
- How satisfied is the team with the internal process? Do they experience any particular difficulties?
- Note for moderator: Make sure that the team take pictures of all whiteboard sketches

D.2 Additional statistics for group assignment

This section presents additional descriptive statistics used for the assignment of project teams to one of the two study groups, i.e. decision view group or comparison group.

Table D.1: *Estimation of project difficulties (CaseVar6, Likert-scale 1: very simple; 5: very difficult)*

Rater	Alpha	Beta	Gamma	Delta
Lecturer 1	2			2
Lecturer 2	4			3
Lecturer 3		4	3	
Lecturer 4		4	3	
Researchers	3	3	2	3
Median	3	4	3	3

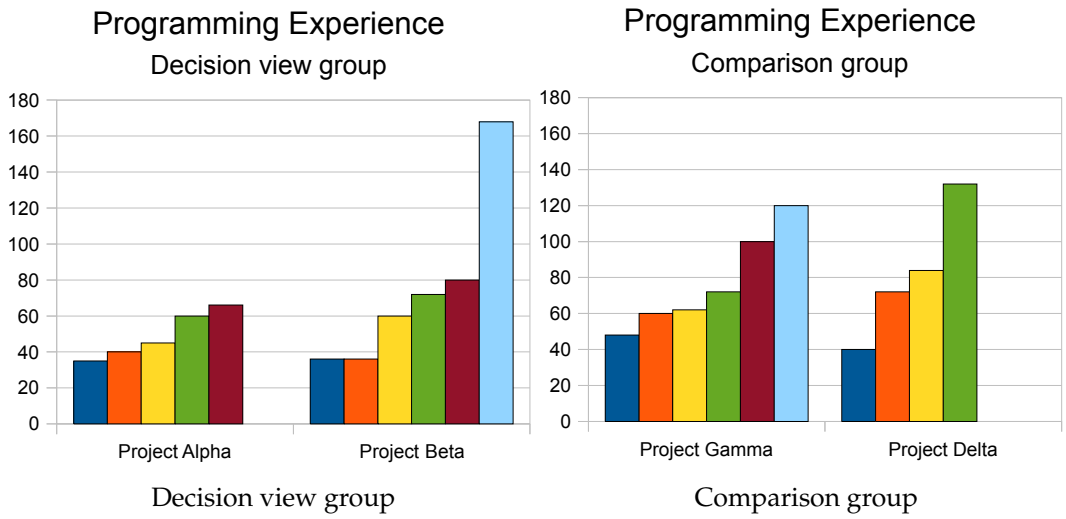


Figure D.1: *Programming experience of the project members in both study groups (CaseVar2)*

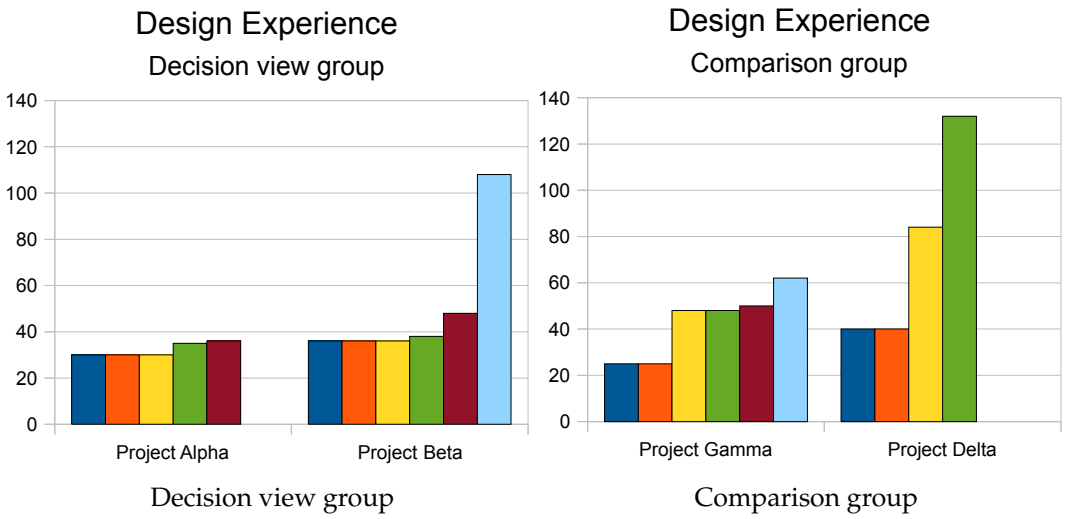


Figure D.2: *Design experience of the project members in both study groups (CaseVar3)*

D.3 Initial visions of the architectures

Figures D.5 and D.6 show examples of early architecture sketches created by the two project teams in the decision view group.

Figure D.3: *Working experience of the project members in both study groups (CaseVar4)*

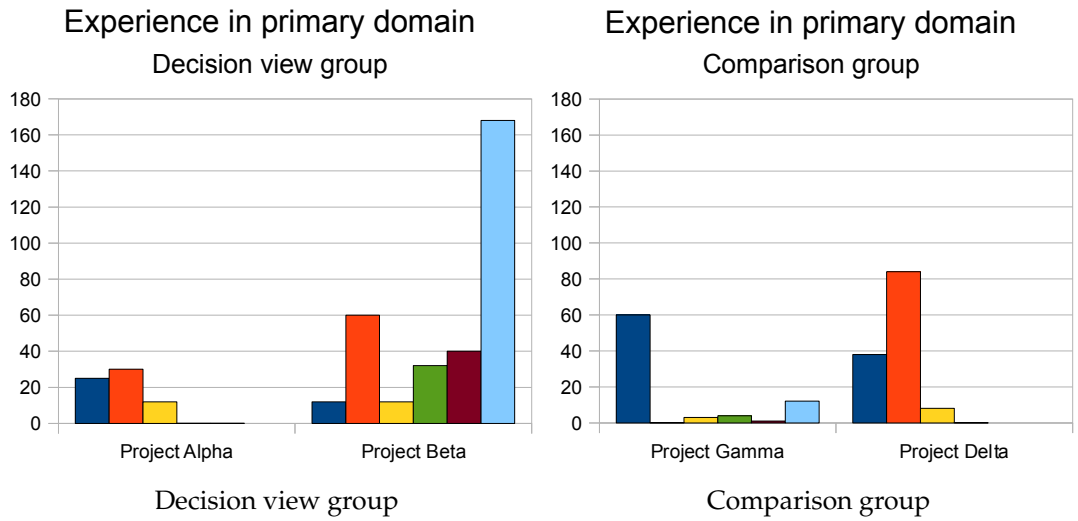


Figure D.4: *Experience in the primary domain of the project members in both study groups (CaseVar7)*

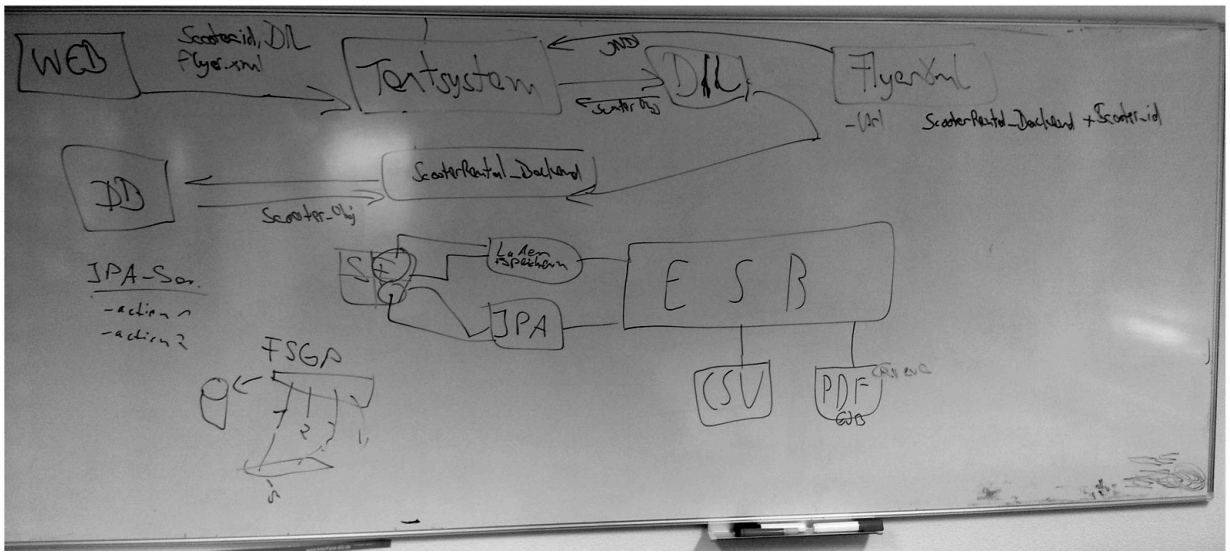


Figure D.5: *Overall architecture envisioned by project team alpha*

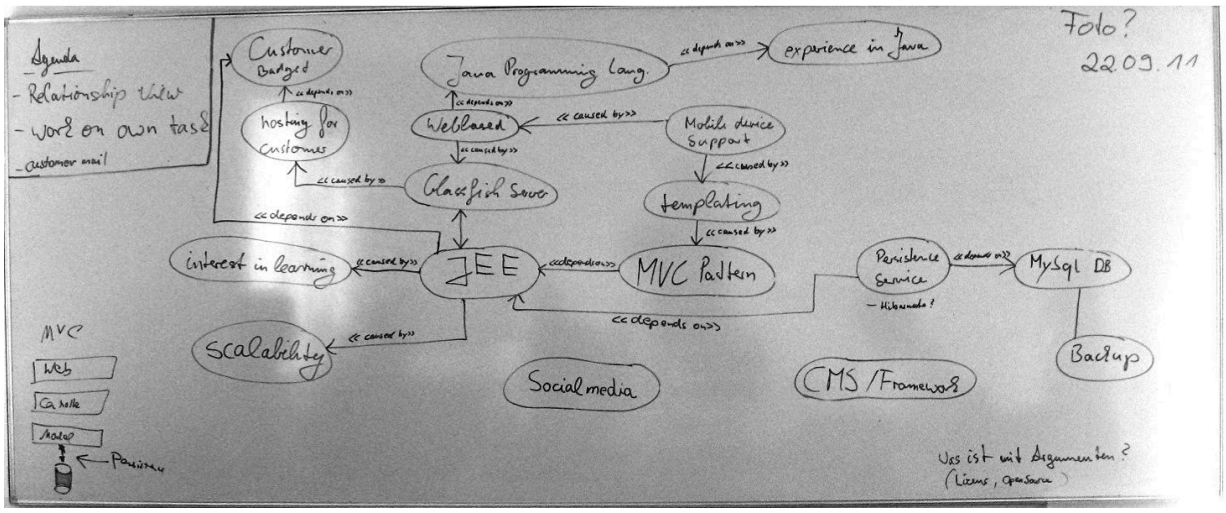


Figure D.6: *Early vision of the architecture created by project team beta (partially using the relationship view*

Bibliography

Adolph, S., Hall, W. and Kruchten, P.: 2011, Using grounded theory to study the experience of software development, *Empirical Software Engineering* **16**(4), 487–513.

Alexander, C.: 1979, *The timeless way of building*, Oxford University Press.

Amazon.com Inc.: 2012, Amazon.com, `http://www.amazon.com`.

Avgeriou, P., Grundy, J., Hall, J., Lago, P. and Mistrik, I.: 2011, *Relating Software Requirements and Architectures*, Springer Publishing Company.

Avgeriou, P. and Zdun, U.: 2005, Architectural patterns revisited – a pattern language, *10th European Conference on Pattern Languages of Programs (EuroPlop), Irsee*.

Avison, D., Lau, F., Myers, M. and Nielsen, P.: 1999, Action research, *Communications of the ACM* **42**(1), 94–97.

Babar, M., Bass, L. and Gorton, I.: 2007, Factors influencing industrial practices of software architecture evaluation: an empirical investigation, *Proceedings of the Quality of software architectures 3rd international conference on Software architectures, components, and applications*, Springer Publishing Company, pp. 90–107.

Babar, M., Dingsyr, T., Lago, P. and van Vliet, H.: 2009, *Software Architecture Knowledge Management: Theory and Practice*, Springer Publishing Company.

Babar, M. and Gorton, I.: 2007, A tool for managing software architecture knowledge, *Proceedings of the Second Workshop on SHAring and Reusing architectural Knowledge Architecture, Rationale, and Design Intent*, IEEE Computer Society, p. 11.

Basili, V. R., Caldiera, G. and Rombach, H. D.: 1994, The goal question metric approach, *Encyclopedia of Software Engineering*, John Wiley & Sons, Inc.

Bass, L., Clements, P. and Kazman, R.: 2003, *Software Architecture in Practice*, second edn, Addison-Wesley.

Bass, L. and Nord, R.: 2012, Understanding the Context of Architecture Evaluation Methods, *Proceedings of the Joint 10th Working IEEE/IFIP Conference on Software Architecture & 6th European Conference on Software Architecture*, IEEE, pp. xx–xx.

Boehm, B., Rombach, H. and Zelkowitz, M.: 2005, *Foundations of Empirical Software Engineering: The Legacy of Victor R. Basili*, Springer Publishing Company.

Bosch, J.: 2004, Software architecture: The next step, *in* F. Oquendo, B. Warboys and R. Morrison (eds), *Software Architecture*, Vol. 3047 of *Lecture Notes in Computer Science*, Springer Publishing Company, pp. 194–199.

Bosch, J. and Molin, P.: 1999, Software architecture design: evaluation and transformation, *IEEE Conference and Workshop on Engineering of Computer-Based Systems, 1999. Proceedings. ECBS'99.*, IEEE, pp. 4–10.

Brereton, P., Kitchenham, B., Budgen, D. and Li, Z.: 2008, Using a protocol template for case study planning, *Proceedings of the 12th international conference on Evaluation and Assessment in Software Engineering*, British Computer Society, pp. 41–48.

Brooks, F.: 2010, *The Design of Design: Essays from a Computer Scientist*, Addison-Wesley.

Bu, W., Tang, A. and Han, J.: 2009, An analysis of decision-centric architectural design approaches, *Proceedings of the 2009 ICSE Workshop on Sharing and Reusing Architectural Knowledge*, IEEE Computer Society, pp. 33–40.

Buschmann, F., Henney, K. and Schmidt, D.: 2007, *Pattern-oriented software architecture: On patterns and pattern languages*, Vol. 5, John Wiley & Sons, Inc.

Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P. and Stal, M.: 1996, *Pattern-oriented software architecture: a system of patterns*, John Wiley & Sons, Inc.

Capilla, R., Nava, F. and Duenas, J.: 2007, Modeling and Documenting the Evolution of Architectural Design Decisions, *Proceedings of the Second Workshop on SHAring and Reusing architectural Knowledge Architecture, Rationale, and Design Intent*, IEEE Computer Society, pp. 9–15.

Carver, J., Jaccheri, L., Morasca, S. and Shull, F.: 2010, A checklist for integrating student empirical studies with research and teaching goals, *Empirical Software Engineering* **15**(1), 35–59.

Ciolkowski, M., Laitenberger, O., Vegas, S. and Biffl, S.: 2003, Practical experiences in the design and conduct of surveys in empirical software engineering, *Empirical Methods and Studies in Software Engineering* pp. 104–128.

Clements, P., Bachmann, F., Bass, L., Garlan, D., Ivers, J., Little, R., Merson, P. and Nord, R.: 2010, *Documenting Software Architectures: Views and Beyond*, 2nd edn, Addison-Wesley.

Clements, P., Kazman, R., Klein, M., Devesh, D., Reddy, S. and Verma, P.: 2007, The duties, skills, and knowledge of software architects, *Proceedings of the Sixth Working IEEE/IFIP Conference on Software Architecture*, IEEE Computer Society, pp. 20–23.

Clerc, V.: 2011, *Architectural Knowledge Management in Global Software Development*, PhD thesis, Vrije Universiteit Amsterdam, The Netherlands.

Clerc, V., Lago, P. and van Vliet, H.: 2007, The architect's mindset, *Proceedings of the Quality of software architectures 3rd international conference on Software architectures, components, and applications*, Springer Publishing Company, pp. 231–249.

Cook, T. and Campbell, D.: 1979, *Quasi-experimentation: Design & Analysis issues for field settings*, Houghton Mifflin Harcourt.

Corbin, J. and Strauss, A.: 2008, *Basics of qualitative research: Techniques and procedures for developing grounded theory*, Sage Publications, Inc.

Creswell, J. and Miller, D.: 2000, Determining validity in qualitative inquiry, *Theory into practice* **39**(3), 124–130.

Cross, N.: 2001, Design cognition: Results from protocol and other empirical studies of design activity, *Design Knowing and Learning: Cognition in Design Education* pp. 79–103.

Cross, N.: 2004, Expertise in design: an overview, *Design Studies* **25**(5), 427–441.

Curtis, B., Krasner, H. and Iscoe, N.: 1988, A field study of the software design process for large systems, *Communications of the ACM* **31**(11), 1268–1287.

Dobrica, L. and Niemela, E.: 2002, A survey on software architecture analysis methods, *IEEE Transactions on Software Engineering* **28**(7), 638–653.

Duenas, J. and Capilla, R.: 2005, The decision view of software architecture, *in* R. Morrison and F. Oquendo (eds), *Software Architecture*, Vol. 3527 of *Lecture Notes in Computer Science*, Springer Publishing Company, pp. 88–126.

Easterbrook, S., Singer, J., Storey, M. and Damian, D.: 2008, Selecting empirical methods for software engineering research, *Guide to advanced empirical software engineering* pp. 285–311.

Epley, N. and Gilovich, T.: 2006, The anchoring-and-adjustment heuristic Why the adjustments are insufficient, *Psychological Science* **17**(4), 311–318.

Falessi, D., Babar, M., Cantone, G. and Kruchten, P.: 2010, Applying empirical software engineering to software architecture: challenges and lessons learned, *Empirical Software Engineering* **15**(3), 250–276.

Farenhorst, R. and de Boer, R.: 2009, *Architectural Knowledge Management: Supporting Architects and Auditors*, PhD thesis, Vrije Universiteit Amsterdam, The Netherlands.

Farenhorst, R. and van Vliet, H.: 2009, Understanding how to support architects in sharing knowledge, *Proceedings of the 2009 ICSE Workshop on Sharing and Reusing Architectural Knowledge*, IEEE Computer Society, pp. 17–24.

Gamma, E., Helm, R., Johnson, R. and Vlissides, J.: 1995, *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley.

Garlan, D., Monroe, R. and Wile, D.: 1997, Acme: an architecture description interchange language, *Proceedings of the 1997 conference of the Centre for Advanced Studies on Collaborative research*, IBM Press, pp. 7–22.

Given, L.: 2008, *The Sage encyclopedia of qualitative research methods*, Vol. 2, Sage Publications, Inc.

Glaser, B.: 1965, The constant comparative method of qualitative analysis, *Social problems* **12**(4), 436–445.

Glaser, B.: 1998, *Doing grounded theory: Issues and discussions*, Sociology Press, Mill Valley, California.

Glaser, B. and Strauss, A.: 1967, *The Discovery of Grounded Theory: Strategies for Qualitative Research.*, New York. Aldine Publishing.

Goldstein, G. and Hersen, M.: 2000, *Handbook of psychological assessment*, 3rd edn, Elsevier Science Inc.

Gotel, O. and Finkelstein, C.: 1994, An analysis of the requirements traceability problem, *Proceedings of the First International Conference on Requirements Engineering*, IEEE, pp. 94–101.

Gray, D.: 2009, *Doing research in the real world*, Sage Publications, Inc.

Harrison, N. and Avgeriou, P.: 2008, Analysis of architecture pattern usage in legacy system architecture documentation, *Proceedings of the Seventh Working IEEE/IFIP Conference on Software Architecture (WICSA 2008)*, IEEE Computer Society, pp. 147–156.

Harrison, N. and Avgeriou, P.: 2011, Pattern-based architecture reviews, *IEEE Software* **28**(6), 66–71.

Harrison, N., Avgeriou, P. and Zdun, U.: 2007, Using patterns to capture architectural decisions, *IEEE Software* **24**(4), 38–45.

Hester, S., Parnas, D. and Utter, D.: 1981, Using documentation as a software design medium, *Bell System Technical Journal* **60**(8), 1941–1977.

Hevner, A.: 2007, A three cycle view of design science research, *Scandinavian Journal of Information Systems* **19**(2), 87–92.

Hevner, A., March, S., Park, J. and Ram, S.: 2004, Design science in information systems research, *MIS Quarterly* **28**(1), 75–105.

Hillside Europe e.V.: 2009, European Conference on Pattern Languages of Programs, `http://hillside.net/europlop/europlop2009/`.

Hofmeister, C., Kruchten, P., Nord, R., Obbink, H., Ran, A. and America, P.: 2007, A general model of software architecture design derived from five industrial approaches, *Journal of Systems and Software* **80**(1), 106–126.

Hofmeister, C., Nord, R. and Soni, D.: 2009, *Applied Software Architecture*, Addison-Wesley.

Hoorn, J., Farenhorst, R., Lago, P. and van Vliet, H.: 2011, The lonesome architect, *Journal of Systems and Software* **84**(9), 1424–1435.

Höst, M. and Runeson, P.: 2007, Checklists for software engineering case study research, *Proceedings of the First International Symposium on Empirical Software Engineering and Measurement*, IEEE Computer Society, pp. 479–481.

IEEE: 2000, *IEEE Std 1471–2000, IEEE Recommended Practice for Architectural Description of Software-Intensive Systems*, IEEE, New York, NY, USA.

IEEE: 2008, *IEEE STD 1028-2008, IEEE Standard for Software Reviews and Audits*, IEEE, New York, NY, USA.

ISO/IEC/IEEE: 2011, *ISO/IEC/IEEE 42010, Systems and software engineering — Architecture description*, ISO/IEC/IEEE.

Jansen, A.: 2008, *Architectural design decisions*, PhD thesis, University of Groningen, The Netherlands.

Jansen, A., Avgeriou, P. and van der Ven, J.: 2009, Enriching software architecture documentation, *Journal of Systems and Software* **82**(8), 1232–1248.

Jansen, A. and Bosch, J.: 2005, Software Architecture as a Set of Architectural Design Decisions, *Proceedings of the 5th Working IEEE/IFIP Conference on Software Architecture*, IEEE Computer Society, pp. 109–120.

Jansen, A., Bosch, J. and Avgeriou, P.: 2008, Documenting after the fact: Recovering architectural design decisions, *Journal of Systems and Software* **81**(4), 536–557.

Jansen, A., de Vries, T., Avgeriou, P. and van Veelen, M.: 2008, Sharing the architectural knowledge of quantitative analysis, *in* S. Becker, F. Plasil and R. Reussner (eds), *Quality of Software Architectures. Models and Architectures*, Vol. 5281 of *Lecture Notes in Computer Science*, Springer Publishing Company, pp. 220–234.

Jansen, A., van der Ven, J., Avgeriou, P. and Hammer, D.: 2007, Tool support for architectural decisions, *Proceedings of the Sixth Working IEEE/IFIP Conference on Software Architecture*, IEEE Computer Society, pp. 4–14.

JBoss.org: 2012, Community driven open source middleware, `http://www.jboss.org/`.

Jedlitschka, A. and Pfahl, D.: 2005, Reporting guidelines for controlled experiments in software engineering, *International Symposium on Empirical Software Engineering, 2005*, IEEE, pp. 92–101.

Jiang, L., Eberlein, A., Far, B. and Mousavi, M.: 2008, A methodology for the selection of requirements engineering techniques, *Software and Systems Modeling* **7**(3), 303–328.

Kazman, R., Bass, L., Webb, M. and Abowd, G.: 1994, Saam: A method for analyzing the properties of software architectures, *Proceedings of the 16th international conference on Software engineering*, IEEE Computer Society, pp. 81–90.

Kazman, R. and Carrière, S.: 1999, Playing detective: Reconstructing software architecture from available evidence, *Automated Software Engineering* **6**(2), 107–138.

Kazman, R., Klein, M. and Clements, P.: 2000, Atam: Method for architecture evaluation, *Technical report*, Software Engineering Institute, Carnegie Mellon University.
URL: *http://www.sei.cmu.edu/publications/documents/00.reports/00tr004.html*

Kitchenham, B., Pfleeger, S., Pickard, L., Jones, P., Hoaglin, D., Emam, K. and Rosenberg, J.: 2002, Preliminary guidelines for empirical research in software engineering, *IEEE Transactions on Software Engineering* **28**(8), 721–734.

Kitchenham, B., Pickard, L. and Pfleeger, S.: 1995, Case studies for method and tool evaluation, *IEEE Software* **12**(4), 52–62.

Kontio, J., Bragge, J. and Lehtola, L.: 2008, The focus group method as an empirical tool in software engineering, *Guide to advanced empirical software engineering* pp. 93–116.

Koschke, R.: 2009, Architecture reconstruction, *Software Engineering*, Springer Publishing Company, pp. 140–173.

Krikhaar, R., Postma, A., Sellink, A., Stroucken, M. and Verhoef, C.: 1999, A two-phase process for software architecture improvement, *Proceedings of the IEEE International Conference on Software Maintenance*, IEEE Computer Society, p. 371.

Kruchten, P.: 1995, The 4+ 1 View Model of Architecture, *IEEE Software* **12**(6), 42–50.

Kruchten, P.: 1999, The software architect, and the software architecture team, *Proceedings of the first IFIP Conference on Software Architecture (WICSA1)*, Kluwer Academic Publishers, pp. 565–583.

Kruchten, P.: 2004a, An ontology of architectural design decisions in software intensive systems, *Proceedings of the 2nd Groningen Workshop on Software Variability*, pp. 54–61.

Kruchten, P.: 2004b, *The Rational Unified Process: an introduction*, Addison-Wesley.

Kruchten, P.: 2008, Controversy corner: What do software architects really do?, *Journal of Systems and Software* **81**(12), 2413–2416.

Kruchten, P., Capilla, R. and Dueñas, J.: 2009, The Decision View's Role in Software Architecture Practice, *IEEE Software* **26**(2), 36–42.

Kruchten, P., Lago, P. and van Vliet, H.: 2006, Building up and reasoning about architectural knowledge, *in* C. Hofmeister, I. Crnkovic and R. Reussner (eds), *Quality of Software Architectures*, Vol. 4214 of *Lecture Notes in Computer Science*, Springer Publishing Company, pp. 43–58.

Lee, L. and Kruchten, P.: 2008, A Tool to Visualize Architectural Design Decisions, *Proceedings of the 4th International Conference on Quality of Software-Architectures: Models and Architectures*, Springer Publishing Company, pp. 43–54.

Lethbridge, T., Sim, S. and Singer, J.: 2005, Studying software engineers: Data collection techniques for software field studies, *Empirical Software Engineering* **10**(3), 311–341.

Liang, P. and Avgeriou, P.: 2009, Tools and Technologies for Architecture Knowledge Management, *Software Architecture Knowledge Management: Theory and Practice*, Springer Publishing Company, pp. 91–111.

Liang, P., Jansen, A. and Avgeriou, P.: 2009, Knowledge architect: A tool suite for managing software architecture knowledge, *Technical Report RUG-SEARCH-09-L01*, SEARCH Group, University of Groningen, The Netherlands.

Liu, J.: 2002, Research Project: An Analysis of JBoss Architecture, `http://www.huihoo.org/jboss/jboss.html`.

Mack, N., Woodsong, C., MacQueen, K., Guest, G. and Namey, E.: 2005, *Qualitative research methods: A data collector's field guide*, FLI.

MacLean, A., Young, R., Bellotti, V. and Moran, T.: 1991, Questions, options, and criteria: elements of design space analysis, *Human-Computer Interaction* **6**(3), 201–250.

Malavolta, I., Muccini, H. and Rekha, V.: 2011, Supporting architectural design decisions evolution through model driven engineering, *Proceedings of the Third international conference on Software engineering for resilient systems*, Springer Publishing Company, pp. 63–77.

Mannion, M. and Keepence, B.: 1995, SMART requirements, *ACM SIGSOFT Software Engineering Notes* **20**(2), 42–47.

March, S. and Smith, G.: 1995, Design and natural science research on information technology, *Decision Support Systems* **15**(4), 251–266.

Muller, G.: 2004, *CAFCR: A Multi-view Method for Embedded Systems Architecting. Balancing Genericity and Specificity*, PhD thesis, Technische Universiteit Delft.

Mustapic, G., Wall, A., Norstrom, C., Crnkovic, I., Sandstrom, K., Froberg, J. and Andersson, J.: 2004, Real world influences on software architecture-interviews with industrial system experts, *Fourth Working IEEE/IFIP Conference on Software Architecture, 2004. WICSA 2004*, IEEE, pp. 101–111.

Nagappan, N., Maximilien, E., Bhat, T. and Williams, L.: 2008, Realizing quality improvement through test driven development: results and experiences of four industrial teams, *Empirical Software Engineering* **13**(3), 289–302.

Nuseibeh, B.: 2001, Weaving Together Requirements and Architectures, *Computer* **34**(3), 115–117.

O'Gorman, T.: 2004, *Applied adaptive statistical methods: tests of significance and confidence intervals*, Society for Industrial Mathematics.

Oracle Corporation: 2002, Core J2EE patterns, `http://java.sun.com/blueprints/corej2eepatterns/`.

Parnas, D.: 2009, Document based rational software development, *Knowledge-Based Systems* **22**(3), 132–141.

Parnas, D. and Clements, P.: 1986, A rational design process: How and why to fake it, *IEEE Transactions on Software Engineering* **12**(2), 251–257.

Parnas, D. L.: 2011, Precise documentation: The key to better software, *in* S. Nanz (ed.), *The Future of Software Engineering*, Springer Publishing Company, pp. 125–148.

Patton, M.: 2002, *Qualitative research and evaluation methods*, Sage Publications, Inc.

Perry, D., Porter, A. and Votta, L.: 2000, Empirical studies of software engineering: a roadmap, *Proceedings of the Conference on the Future of Software Engineering*, ACM, pp. 345–355.

Perry, D. and Wolf, A.: 1992, Foundations for the study of software architecture, *ACM SIGSOFT Software Engineering Notes* **17**(4), 40–52.

Purcell, A. and Gero, J.: 1998, Drawings and the design process:: A review of protocol studies in design and other disciplines and related research in cognitive psychology, *Design studies* **19**(4), 389–430.

Ramesh, B. and Jarke, M.: 2001, Toward reference models for requirements traceability, *IEEE Transactions on Software Engineering* **27**(1), 58–93.

Robson, C.: 2011, *Real world research*, John Wiley & Sons, Inc.

Rossow, C., Dietrich, C., Bos, H., Cavallaro, L., van Steen, M., Freiling, F. and Pohlmann, N.: 2011, Sandnet: Network traffic analysis of malicious software, *Proceedings of the First Workshop on Building Analysis Datasets and Gathering Experience Returns for Security*, ACM, pp. 78–88.

Rozanski, N. and Woods, E.: 2005, *Software systems architecture: working with stakeholders using viewpoints and perspectives*, Addison-Wesley.

Runeson, P. and Höst, M.: 2009, Guidelines for conducting and reporting case study research in software engineering, *Empirical Software Engineering* **14**, 131–164.

Schmidt, D. and Buschmann, F.: 2003, Patterns, frameworks, and middleware: their synergistic relationships, *Proceedings of the 25th International Conference on Software Engineering*, IEEE Computer Society, pp. 694–704.

Schwaber, K. and Beedle, M.: 2002, *Agile software development with Scrum*, Prentice-Hall.

Seaman, C.: 1999, Qualitative Methods in Empirical Studies of Software Engineering, *IEEE Transactions on Software Engineering* **25**(4), 557–572.

Serral, E., Valderas, P. and Pelechano, V.: 2010, Towards the model driven development of context-aware pervasive systems, *Pervasive and Mobile Computing* **6**(2), 254–280.

Shahin, M., Liang, P. and Khayyambashi, M.: 2009, Architectural design decision: Existing models and tools, *European Conference on Software Architecture. WICSA/ECSA 2009. Joint Working IEEE/IFIP Conference on Software Architecture, 2009*, IEEE, pp. 293–296.

Shahin, M., Liang, P. and Khayyambashi, M.: 2010, Improving understandability of architecture design through visualization of architectural design decision, *Proceedings of the 2010 ICSE Workshop on Sharing and Reusing Architectural Knowledge*, ACM, pp. 88–95.

SHARK Workshop: 2012, Workshop on SHAring and Reusing architectural Knowledge, `http://www.shark-workshop.org/`.

Sharp, H., DeSouza, C. and Dittrich, Y.: 2010, Using ethnographic methods in software engineering research, *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 2*, ACM, pp. 491–492.

Shaw, M.: 2002, What makes good research in software engineering?, *International Journal on Software Tools for Technology Transfer (STTT)* **4**(1), 1–7.

Shull, F., Singer, J. and Sjøberg, D.: 2008, *Guide to Advanced Empirical Software Engineering*, Springer Publishing Company.

Sjoberg, D., Dyba, T. and Jorgensen, M.: 2007, The future of empirical methods in software engineering research, *Future of Software Engineering, 2007. FOSE'07*, IEEE, pp. 358–378.

Software, E.: 2012, Trac, `http://trac.edgewall.org/`.

Sonnentag, S.: 1998, Expertise in professional software design: A process study, *Journal of Applied Psychology* **3**(5), 703–715.

Stake, R.: 1995, *The art of case study research*, Sage Publications, Inc.

Stevens, S.: 1946, On the theory of scales of measurement, *Science* **103**(2684), 677–680.

Strauss, A.: 1987, *Qualitative analysis for social scientists*, Cambridge Univ Pr.

Svahnberg, M., Aurum, A. and Wohlin, C.: 2008, Using students as subjects-an empirical evaluation, *Proceedings of the Second ACM-IEEE international symposium on Empirical software engineering and measurement*, ACM, pp. 288–290.

Tang, A., Aleti, A., Burge, J. and van Vliet, H.: 2010, What makes software design effective?, *Design Studies* **31**(6), 614–640.

Tang, A., Avgeriou, P., Jansen, A., Capilla, R. and Ali Babar, M.: 2010, A comparative study of architecture knowledge management tools, *Journal of Systems and Software* **83**(3), 352–370.

Tang, A., Babar, M., Gorton, I. and Han, J.: 2006, A survey of architecture design rationale, *Journal of Systems and Software* **79**(12), 1792–1804.

Tang, A., Jin, Y. and Han, J.: 2007, A rationale-based architecture model for design traceability and reasoning, *Journal of Systems and Software* **80**(6), 918–934.

Tang, A. and Lago, P.: 2010, Notes on design reasoning tactics, *Technical Report SUTICT-TR2010.01*, Swinburne University of Technology.
URL: *http://www.swinburne.edu.au/ict/research/documents/SUTICT_TR2010_01.PDF*

Tang, A., Tran, M., Han, J. and Van Vliet, H.: 2008, Design reasoning improves software design quality, *Quality of Software Architectures. Models and Architectures* pp. 28–42.

Taylor, R. N., Medvidovic, N. and Dashofy, E. M.: 2009, *Software Architecture: Foundations, Theory, and Practice*, first edn, John Wiley & Sons, Inc.

Tigris.org: 2012, Subversion, `http://subversion.tigris.org`.

Trochim, W.: 2001, *Research methods knowledge base*, Atomic Dog Publishing.

Tyree, J. and Akerman, A.: 2005, Architecture Decisions: Demystifying Architecture, *IEEE Software* **22**(2), 19–27.

University of Groningen, Software Engineering and Architecture Group: 2012a, The Open Decision Repository, `http://opendecisionrepository.googlecode.com`.

University of Groningen, Software Engineering and Architecture Group: 2012b, The Open Pattern Repository, `http://code.google.com/p/openpatternrepository/`.

Urquhart, C., Lehmann, H. and Myers, M.: 2010, Putting the theory back into grounded theory: guidelines for grounded theory studies in information systems, *Information systems journal* **20**(4), 357–381.

van der Ven, J., Jansen, A., Avgeriou, P. and Hammer, D.: 2006, Using architectural decisions, *Second International Conference on the Quality of Software Architecture (Qosa 2006)*.

van der Ven, J., Jansen, A., Nijhuis, J. and Bosch, J.: 2006, *Design decisions: The bridge between rationale and architecture*, Springer Publishing Company, pp. 329–348.

van Heesch, U. and Avgeriou, P.: 2009, A pattern driven approach against architectural knowledge vaporization, *Proceedings of the 14th European Conference on Pattern Languages of Programs (EuroPLoP), Irsee*.

van Heesch, U. and Avgeriou, P.: 2010, Naive architecting-understanding the reasoning process of students: a descriptive survey, *Proceedings of the 4th European conference on Software architecture*, Springer Publishing Company, pp. 24–37.

van Heesch, U. and Avgeriou, P.: 2011, Mature Architecting - A Survey about the Reasoning Process of Professional Architects, *Proceedings of the 2011 Ninth Working IEEE/IFIP Conference on Software Architecture*, IEEE Computer Society, pp. 260–269.

van Heesch, U., Avgeriou, P. and Hilliard, R.: 2012a, A documentation framework for architecture decisions, *Journal of Systems and Software* **85**(4), 795–820.

van Heesch, U., Avgeriou, P. and Hilliard, R.: 2012b, Forces on Architecture Decisions - A Viewpoint, *Proceedings of the Joint 10th Working IEEE/IFIP Conference on Software Architecture & 6th European Conference on Software Architecture*, IEEE, pp. xx–xx.

van Heesch, U., Avgeriou, P., Zdun, U. and Harrison, N.: 2012, The supportive effect of patterns in architecture decision recovery— a controlled experiment, *Science of Computer Programming* **77**(5), 551 – 576.

Van Lamsweerde, A.: 2001, Goal-Oriented Requirements Engineering: A Guided Tour, *Proceedings of the Fifth IEEE International Symposium on Requirements Engineering*, IEEE Computer Society, p. 249.

Verner, J., Sampson, J., Tosic, V., Bakar, N. and Kitchenham, B.: 2009, Guidelines for industrially-based multiple case studies in software engineering, *Third International Conference on Research Challenges in Information Science. RCIS 2009*, IEEE, pp. 313–324.

Wieringa, R.: 2009, Design science as nested problem solving, *Proceedings of the 4th international conference on design science research in information systems and technology*, ACM, pp. 8–20.

Williams, L. and Smith, C.: 2002, Pasa sm: a method for the performance assessment of software architectures, *Proceedings of the 3rd International Workshop on Software and Performance*, ACM, pp. 179–189.

Williams, R., Pandelios, G. and Behrens, S.: 1999, Software Risk Evaluation (SRE) Method Description (Version 2.0), *Technical Report CMU/SEI-99-TR-029, ESC-TR-99-029*, Software Engineering Institute, Carnegie Mellon University.

Wohlin, C., Höst, M. and Henningsson, K.: 2003, Empirical research methods in software engineering, *Empirical Methods and Studies in Software Engineering* pp. 7–23.

Wohlin, C., Runeson, P., Höst, M., Ohlsson, M., Regnell, B. and Wesslén, A.: 2012, *Experimentation in software engineering*, Springer Publishing Company.

Yan, H., Garlan, D., Schmerl, B., Aldrich, J. and Kazman, R.: 2004, Discotect: A system for discovering architectures from running systems, *Proceedings of the 26th International Conference on Software Engineering*, IEEE Computer Society, pp. 470–479.

Yin, D. R. K.: 2003, *Case Study Research: Design and Methods, Applied Social Research Methods Series, Vol 5*, third edition edn, Sage Publications, Inc.

Zannier, C., Chiasson, M. and Maurer, F.: 2007, A model of design decision making based on empirical results of interviews with software designers, *Information and Software Technology* **49**(6), 637–653.

Zdun, U.: 2007, Systematic pattern selection using pattern language grammars and design space analysis, *Software-Practice & Experience* **37**(9), 983–1016.

Zimmermann, O., Grundler, J., Tai, S. and Leymann, F.: 2007, Architectural decisions and patterns for transactional workflows in soa, *Proceedings of the 5th international conference on Service-Oriented Computing*, Springer Publishing Company, pp. 81–93.

Zimmermann, O., Gschwind, T., Küster, J., Leymann, F. and Schuster, N.: 2007, Reusable architectural decision models for enterprise application development, *Proceedings of the Quality of software architectures 3rd international conference on Software architectures, components, and applications*, Springer Publishing Company, pp. 15–32.

Zimmermann, O., Zdun, U., Gschwind, T. et al.: 2008, Combining pattern languages and reusable architectural decision models into a comprehensive and comprehensible design method, *Proceedings of the Seventh Working IEEE/IFIP Conference on Software Architecture (WICSA 2008)*, IEEE Computer Society, pp. 157–166.