ISBN 978-82-326-1478-3 (printed version)
ISBN 978-82-326-1479-0 (electronic version)
ISSN 1503-8181

Doctoral theses at NTNU, 2016:70

Mohsen Anvaari

A Rule-based Framework for Enhancing Architectural Decision Guidance

NTNU
Norwegian University of
Science and Technology
Faculty of Information Technology,
Mathematics and Electrical Engineering
Department of Computer and Information Science

NTNU
Norwegian University of
Science and Technology

Doctoral theses at NTNU, 2016:70

NTNU

NTNU
Norwegian University of
Science and Technology

Mohsen Anvaari

A Rule-based Framework for Enhancing Architectural Decision Guidance

Thesis for the degree of Philosophiae Doctor

Trondheim, May 2016

Norwegian University of Science and Technology
Faculty of Information Technology,
Mathematics and Electrical Engineering
Department of Computer and Information Science

NTNU
Norwegian University of Science and Technology

Thesis for the degree of Philosophiae Doctor

Faculty of Information Technology,
Mathematics and Electrical Engineering
Department of Computer and Information
Science

ISBN 978-82-326-1478-3 (printed version) ISBN
978-82-326-1479-0 (electronic version) ISSN
1503-8181

Doctoral theses at NTNU, 2016:70

Printed by Skipnes Kommunikasjon as

TO MY FAMILY

Abstract

Architectural decision-making is a non-trivial process for architects in software development projects. In many cases, such a process starts by identifying architectural issues that an architect should make decisions about. In the second step, the architects explore available alternatives to solve the architectural issues. In the final step, the architects choose one of the candidate alternatives for each issue, based on the decision drivers. Notable progress has been made to assist practitioners in choosing one alternative among the possible alternatives. Several methods and tools have been developed for documenting the rationale and the outcome of the decision-making process. There is, nevertheless, little research focusing on identifying architectural issues that are eligible for a particular project. In the absence of systematic methods of identifying architectural issues, practitioners mainly start their architectural decision-making process based on intuition and prior experience, which may be insufficient due to cognitive biases.

We have investigated the industrial context to understand the attitudes and challenges of large-scale enterprises in making and reusing architectural decisions. Then, we have reviewed the literature to identify the gap in developing architectural knowledge about the past into architectural decision guidance for the future. Afterwards, we have tackled the problem of enhancing architectural guidance by developing a framework called Semi Automated Design Guidance Enhancer (SADGE). SADGE extracts architectural issues from project documents and domain literature by applying natural language processing (NLP). This encourages practitioners to identify more architectural issues in the early phases of their projects, making them more prepared for the later phases, when changing and/or refactoring the architecture is more costly.

Finally, we have evaluated the framework by conducting a case study on project documents and running experiments with IT students and expert IT architects. The results of the evaluation show that SADGE extracts architectural issues with a significant recall while reducing the manual knowledge processing effort notably. The evaluation also reveals that the experts believe that the framework can be very helpful for them to either reduce the amount of text to read, or to identify hot spots in their documents that need extra attention.

The main contributions of this thesis are:

C1 An overview of the state-of-the-art and state-of-the-practice in making and reusing architectural decisions.

C2 A rule-based framework for developing architectural knowledge in project documents and domain literature into architectural decision guidance.

C3 Results of empirical evaluation of developing architectural decision guidance by employing a rule-based framework.

Preface

This dissertation is submitted to the Norwegian University of Science and Technology (NTNU) in partial fulfillment of the requirements for the degree philosophiae doctor.

The work contained herein has been performed at the Department of Computer and Information Science, NTNU, Trondheim, under supervision of Professor Reidar Conradi, Doctor Carl-Fredrik Sørensen and Professor Letizia Jaccheri.

The thesis has been supported by the SmartGrids initative at NTNU. It also included a 25% teaching duty at the department of Computer and Information Science.

Acknowledgements

Foremost, I would like to thank Professor Reidar Conradi for his caring supervision during my PhD journey. I would also like to express my appreciation to Dr. Carl-Fredrik Sørensen, who joined the supervision team during the last and most crucial phase of my PhD, and helped me with his thoughts and pushes. Besides, I would like to express my gratitude to Professor Letizia Jaccheri for her support from both the academic and organizational levels. I would like to thank Dr. Daniela S. Cruzes for enlightening me in the beginning of my PhD. I also thank my colleagues at IDI, NTNU for their encouragement and fruitful discussions.

Special thanks to Professor Olaf Zimmermann who inspired me during my stay at the University of Applied Sciences of Eastern Switzerland. After returning back to NTNU, he has still supported me by his insightful feedbacks till the end of this PhD. I also thank companies, who openly gave me data, and the practitioners and students who participated in my studies.

Many thanks to my friends, especially the Husom family, who have always been a source of energy and motivation. Special thanks to my dear Ghazal for supporting me during the final stages of this PhD.

Finally, I would like to thank my family who has been calling for my success as always. My parents, Fatemeh and Ali, for their continuous love, my siblings, Zohreh, Maryam, and Ebrahim for their encouragement and my nieces, Setareh and Bahareh for their positive energy.

Contents

List of Tables

List of Figures

CHAPTER 1

Introduction

This chapter presents the problem statement and the research questions of this thesis. Then it briefly introduces the research methods and the claimed contributions of the thesis. Finally, it outlines the structure of the thesis.

1.1 Problem Statement

All software systems have fundamental structure known as software architecture. The software architecture comprises the components and the connectors of the software, and determines where the components are located, how they are connected, and in what way they communicate with each other [BCK03]. Designing the architecture requires several high- and low-level decisions. Therefore, the software architecture is considered as a result of architectural decision-making processes. The more complicated the software system, the more complex its architecture, and consequently the architectural decision-making process becomes more difficult. Hence, in the companies that are in charge of developing large and distributed software systems, the architectural decision-making process is a crucial and non-trivial endeavor.

In many cases, the decision-making process starts with identifying and recognizing architectural issues[1] that an architect should make decisions about. In the second step, the

[1] Literature calls the architectural issues, *decision topics* [JBA08] or *decisions required* [ZM12] too. We adhere to architectural issues in this thesis.

architects explore and recognize available alternatives for solving the architectural issues. In the last step, for each issue, the architects choose one of the candidate alternatives based on the decision drivers (mainly quality attributes such as security, performance, reliability, etc. and business drivers such as time-to-market and cost) [FCKK11].

For example, in designing the presentation layer of a web application, one of the architectural issues to make a decision about, is *input and data validation strategy*. The alternative solutions are *accept known good, reject known bad,* and *sanitize* [Net09]. A possible decision (outcome of the decision-making process) is choosing accept known good strategy based on *security, reliability, performance*, and *usability* as the main decision drivers.

In the software architecture research community, notable progress has been made to assist practitioners in choosing and evaluating one alternative among candidate alternatives (e.g. Architecture Tradeoff Analysis Method and Cost Benefit Analysis Method). Several methods and tools have been developed for documenting the rationale and the outcome of the decision-making process [BDLvV09, TAJ+10]. Little work, nevertheless, have focused on identifying architectural issues in a specific project. This is the first and very important step of the decision-making process. Observations show that architects often do not identify architectural issues based on literature studies, or systematic reuse of knowledge already gained [Zim09]. Rather, they mostly rely on their intuitions and prior experiences to recognize architectural issues [Kru13, Zim09] that may be insufficient due to cognitive biases [SM95].

There is a promising approach that has been developed to help practitioners in the first step of their decision-making process. The approach is a decision identification technique that enhances architectural guidance (a list of architectural issues/decisions topics and their alternative solutions) from decisions made in previous projects, and from knowledge about a domain that can be found in the literature. Through manual decision identification rules, this approach tasks a knowledge engineer to study pattern languages and books, technical reports, industrial standards, and project documentation to identify architectural issues [Zim09]. The technique advises knowledge engineers to read the natural language texts of the literature documents and to annotate the texts manually. The intention is to extract architectural issues from documents and to develop architectural decision guidance from the extracted information. Such guidance is a reusable asset containing knowledge about architectural issues recurring in a particular domain [Zim11]. Several case studies have shown that the developed architectural guidance is promising in assisting the practitioners in their decision making, e.g., in SOA design [ZMK12].

The decision identification approach described above is manual; due to the knowledge engineering effort that has to be invested initially, practitioners still are reluctant to use it. For instance, in our trial to annotate the sentences that contain architectural issues out of a 500 pages architectural textbook, we spent 50 hours to apply the manual decision identification approach. Therefore, developing approaches that extract issues from architecture-related documents in a more automatic way, would accelerate the architectural guidance

development, and as a result encourage the practitioners to use such guidance.

This thesis explores the current state of architectural decision making process in companies that develop large and distributed software (e.g., Smart Grid software applications or telecommunication systems). By grounding on the advances in information extraction and natural language processing domains, it develops a rule-based framework called SADGE that enhances architectural guidance from architecture-related documents in a rapid way. The thesis evaluates the efficiency and effectiveness of the framework by conducting a case study on documents received from a telecommunication company, and by performing experiments on both IT students and expert IT/software architects.

1.2 Research Context

This research started as a part of the SmartGrids project at NTNU that was launched in 2011. The initial topic of the research was Improved Management of Software Evolution for Smart Grid Applications, and the intention was to propose and validate methods for developing and evolving the related software systems in electrical power grids. After narrowing the topic to the architectural decision making process in electrical power grids, we found that the electrical grid companies in Norway have not captured many architectural decisions yet; thus, access to enough documentation to start the study was not feasible. Therefore, we expanded the case of study to similar software systems such as those in the telecommunication area. As a result, we consider the target application of this research any large and distributed software system (what we refer to ultra-large-scale system or system-of-systems in this thesis) besides Smart Grid software applications.

1.3 Research Questions

The research questions this thesis addresses are:

RQ1 *What is the attitude of large-scale enterprises in making and reusing architectural decisions and how do available tools and research prototypes support them*?

RQ2 *How can a framework be established to develop architectural decision guidance from architecture-related documents in a rapid way*?

RQ3 *How efficient and effective will such a framework be in developing architectural decision guidance*?

1.4 Research Methods

Several research methods have been used to answer the research questions:

M1 *Qualitative (in-depth) interviews*: Several qualitative interviews have been conducted with informants from the software industry to answer a part of RQ1.

M2 *Survey*: A survey has been conducted on experts to measure their agreement degree in architectural decision-making process. The findings partially answers RQ1.

M3 *Literature review*: A literature review has been conducted to answer another part of RQ1.

M4 *Design science*: A framework has been developed, based on the design science approach or engineering method, to answer RQ2.

M5 *Case study*: A case study has been conducted to improve and evaluate the framework.

M6 *Experiments*: Two experiments have been conducted to evaluate the framework; one with IT students, and one with expert IT/software architects in the industry.

Section 2.7 gives a definition of each of these methods. Chapter 3 clarifies how these methods contribute to answer the research questions.

1.5 Contributions

The contributions of this thesis can be divided into two main themes:

T1 Exploration and investigation of making and reusing architectural decisions in practice and in the literature.

T2 Development and evaluation of a framework that accelerates the identification of architectural issues, and helps architects to make architectural decisions in a more efficient way.

The main contributions are:

C1 An overview of the state-of-the-art and state-of-the-practice in making and reusing architectural decisions.

- C1.1 The attitude of large-scale enterprises in making and reusing architectural decisions and, the effect of SECO (software ecosystem) relationships on their decisions.

C1.2 Perspectives of existing tools and research prototypes that facilitate the post-processing of architectural knowledge from projects and enhancing decision guidance.

C2 SADGE, a rule-based framework for enhancing architectural decision guidance from architectural knowledge in project documents and in the domain literature.

C2.1 A Catalogue of Terms (CoT) needed to automate the extraction of architectural issues from company- or project-specific documents.

C3 Results of empirical evaluation of developing architectural decision guidance by employing a rule-based framework.

1.6 List of Papers

The contribution of this thesis is presented through 5 main papers (P1 to P5) and one supporting paper (P0). This subsection summarizes the papers. The papers are listed historically with the oldest first. Appendix A presents the full content of all of the main papers and Appendix B presents the abstract of the supporting paper.

P0 *Smart Grid Software Applications as an Ultra-Large-Scale System - Challenges for Evolution.*
Mohsen Anvaari, Daniela S. Cruzes, Reidar Conradi.
In Proc. Innovative Smart Grid Technologies (ISGT), 2012 IEEE PES.

P1 *Architectural Decision-Making in Enterprises - Preliminary Findings From an Exploratory Study in Norwegian Electricity Industry.*
Mohsen Anvaari, Reidar Conradi, Letizia Jaccheri.
In Proc. The 7th European Conference on Software Architecture, ECSA 2013.

P2 *Towards Reusing Architectural Knowledge as Design Guides - Functional Requirements, Tool Analysis and Research Roadmap.*
Mohsen Anvaari, Olaf Zimmermann.
In Proc. The 26th International Conference on Software Engineering and Knowledge Engineering, SEKE 2014.

P3 *Semi-automated Design Guidance Enhancer (SADGE): A Framework for Architectural Guidance Development.*
Mohsen Anvaari, Olaf Zimmermann.
In Proc. The 8th European Conference on Software Architecture, ECSA 2014.

P4 *Rule-based Extraction of Architectural Issues from Software Architecture Documents*.

Mohsen Anvaari, Olaf Zimmermann, Carl-Fredrik Sørensen.
Submitted for publication.

P5 *Associating Architectural Issues with Quality Attributes: A Survey on Expert Agreement*.
Mohsen Anvaari, Carl-Fredrik Sørensen, Olaf Zimmermann.
Submitted for publication.

Table 1.1 presents the relationships between research questions, research methods, contributions, and papers in this thesis.

Research Question	Research Method	Contribution	Paper
RQ1	M1, M2, M3	C1	P1, P2, P5
RQ2	M4, M5	C2	P3, P4
RQ3	M5, M6	C3	P3, P4

Table 1.1: Research questions vs. research methods vs. contributions vs. papers

Figure 1.1 positions the research questions and the papers of this thesis in the context of problem-solution/theory-practice landscape[2]. This thesis has covered a cyclic path: Step 1) identifying the problem in practice, Step 2) finding the gap in theory, Step 3) developing a solution on the basis of theory, Step 4) evaluating the solution in practice and Step 5) ending the path by identifying the problem in practice to be tackled next. Chapter 4 is accordingly divided into 5 sections covering the summary of results and contributions of each step.

1.7 Thesis Structure

The thesis is organized into seven chapters and three appendixes as following:

Chapter 1 (this chapter) describes the problem statement and context of the thesis, and outlines the research questions, research methods, and research contributions. Chapter 2 presents state-of-the-art and basic theoretical concepts relevant to the thesis. Chapter 3 contains the research methods used in this thesis. In Chapter 4, the thesis results are presented and analysed. Chapter 5 discusses the research findings against the state-of-the art and argues the research limitations. Chapter 6 as the concluding remarks, summarizes the answers to the research questions. Finally, future work is presented in Chapter 7. Appendix A contains the five selected papers. Appendix B includes a supporting paper (P0). Appendix C and Appendix C contain supporting material.

[2]The figure is inspired by conversations I had with Prof. Olaf Zimmermann during my stay at the Institute for Software at the University of Applied Sciences (HSR FHO) in Rapperswil, Switzerland.

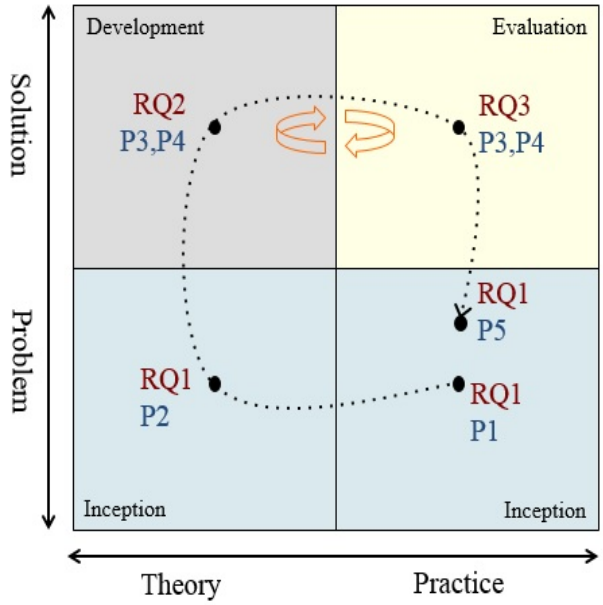


Figure 1.1: The research roadmap within the problem-solution/theory-practice landscape

CHAPTER 2

Theoretical Background

This chapter gives an introduction to the basic concepts and theories relevant for this thesis. Figure 2.1 presents an overview of the goals and context of this study presented in Chapter 1. This figure rationalizes the presence of the concepts and theories that are introduced in this chapter.

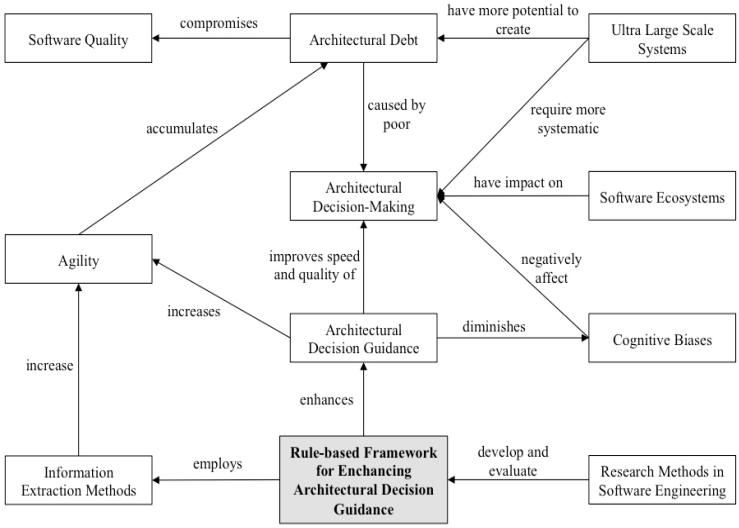


Figure 2.1: The research domain of the study

2.1 Ultra Large Scale Systems

A System-of-Systems (SoS) is a system where its components are large and complex enough to be considered as systems in their own right [Mai98]. SoS is comprised of constituent systems that possesses operationally and managerially independent characteristics [Lan07]. An Ultra-Large-Scale System (ULSS) is a complex software-intensive system that is deeply embedded in a business and social context with many and diverse stakeholders [Sha08]. ULSS is a socio-technical ecosystem and involves people, policies, cultures and economies in addition to large-scale software intensive systems [GKN+07]. Although many authors believe that ULSS is more complex and challenging than SoS [MASS08, Sha08, NFG+06], some of them discuss that ULSS inherits characteristics of SoS, so it has some characteristics in common with today's SoS [NFG+06, RA10].

Maier considers five main characteristics for SoS that distinguish it from very large and complex, but monolithic systems: operational independence of the elements, managerial independence of the elements, evolutionary development rather than fully planned existence, emergent behavior, and geographic distribution [Mai98].

The report from the Software Engineering Institute on ULSS, which is the main reference in ULSS area, claims that characteristics of ULSS that will arise because of their scale are much more revealing; Maier's definitions are not so useful for understanding the underlying technical problems of ULSS [NFG+06]. It considers seven characteristics for ULSS, where some are common with SoS, and others are particular to ULSS:

1. Decentralization in a variety of ways including decentralized data, development, evolution, and operational control.
2. Inherently conflicting, unknowable, and diverse requirements.
3. Continued evolution and deployment rather than staged evolution.
4. Heterogeneous, inconsistent, and changing elements rather than uniform parts.
5. Erosion of the people/system boundary.
6. Normal break-downs rather than excepted ones.
7. New paradigms for acquisition and policy [NFG+06].

Figure 2.2 demonstrates the domains of characteristics in SoS and ULSS. It shows how similar or distinct these two system types are as explained by their characteristics. Examples of ULSS are todays software systems in telecommunication, smart grids, banking, health care, and military information systems.

One of the characteristics of ULSS is decentralized development. The actors that are involved in development of an ULSS are distributed. The development and evolution of an ULSS is the result of interaction between these actors. The actors and their interaction

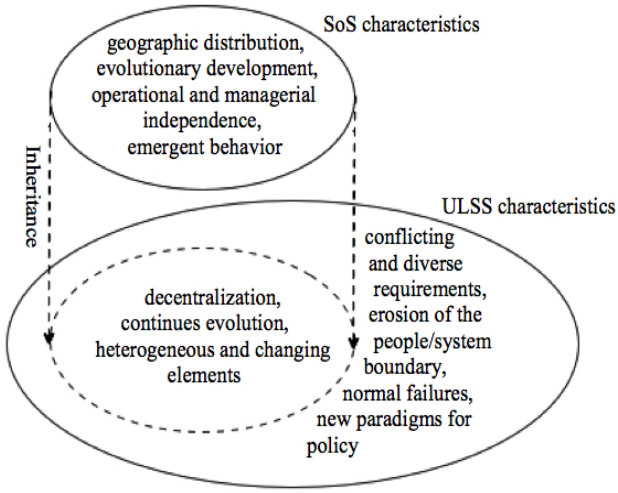


Figure 2.2: SoS and ULSS characteristics domains.

form an ecosystem known as software ecosystem (SECO) in the software engineering domain. In the next section, the SECO concept is elaborated.

2.2 Software Ecosystem

A software ecosystem (SECO) is ”a set of actors functioning as a unit and interacting with a shared market for software and services, together with the relationships among them” [JBF09]. The actor type in a SECO could be a supplier, independent software vendor, software consulting company or intermediary, and a customer [BSJ09]. Interaction or relationship types could be a product flow, service flow, financial flow, or a content flow [BSJ09]. For example the modern grid utilities as a part of a smart grid, purchase and integrate various software applications for launching their enterprise system. The actors in the SECO in this example, are *suppliers* (e.g. Oracle or Microsoft), *independent software vendors* that develop Smart Grid applications, *software consultant companies* that help grid utilities to integrate the applications, and grid utilities as the *customers* of the ecosystem.

There are several visualization techniques to model and illustrate a software supply network (SSN) within a SECO [LBJH12]. To illustrate the SSN of the case in the first step of this PhD research, we have used the model by Brinkkemper et al. [BSJ09].

2.3 Software Quality

There are many definitions of the term quality in the literature. One of the earliest definitions, pointing to the quality of industrial products in general, was given almost a century ago by Radford: "The term *quality*, as applied to the products turned out by industry, means the characteristic or group or combination of characteristics which distinguishes one article from another, or the goods of one manufacturer from those of his competitors, or one grade of product from a certain factory from another grade turned out by the same factory" [Rad22]. Applying the Radford's definition, to develop a definition for software quality, characteristics or group of characteristics that distinguish one software product from another should be identified. In this regard, different academic institutions, standard organizations, and researchers have suggested various characteristics for software quality. The standard ISO/IEC 25010 (originally ISO/IEC 9126) is one of the most adopted standards in the software community that classifies software quality in a structured set of 8 characteristics, 37 sub-characteristics and their attributes [ISO11a]. The characteristics and sub-characteristics of this model are shown in Figure 2.3. Other quality models may include fewer or additional characteristics (such as CISQ [SC13] or McCall [CMRW77]), or may use other synonyms for some of the characteristics (e.g. usability instead of operability, or portability instead of transferability).

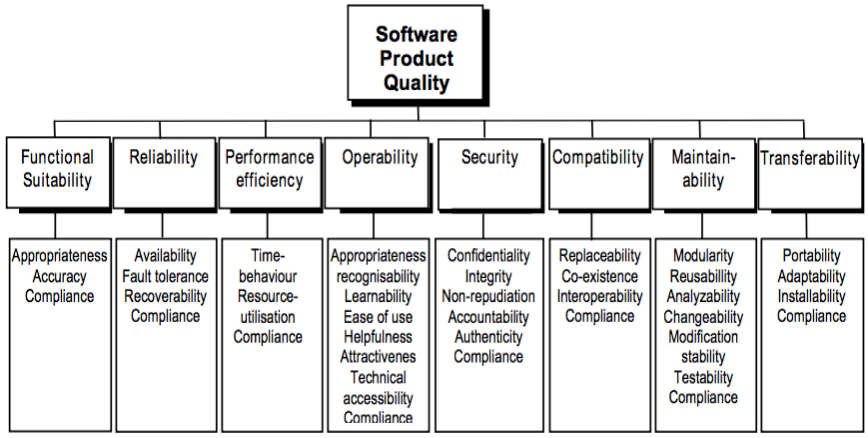


Figure 2.3: Software quality, ISO/IEC 25010 model [ISO11a]

These quality attributes (characteristics) take various roles in different phases of the software development life cycle. In the requirement elicitation phase, they are considered as non-functional requirements. In the architecture and design phase, they are treated as decisions drivers; architects and designers analyze trade-offs between quality attributes when they become conflicting, and satisfying all of them at the same time become impossible. In the test phase, test plans and objectives are developed with regard to the quality attributes.

Satisfying the quality attributes is the overall goal of the software development life cycle, besides fulfilling the required functionality of the software product. Nevertheless, the quality attributes are often compromised due to other stakeholder concerns such as cost and schedule. This phenomenon is elaborated in Section 2.5.2.

2.4 Reuse in Software Engineering

”Software reuse is the process of creating software systems from existing software rather than building software systems from scratch” [Kru92]. The main goal is to reduce cost and save time of development by replacing creation with recycling [JSS14]. Besides, it increases development productivity, software reliability, and ease of maintenance. It also improves the quality of documentation and testing, and increases the speed of replacing aging systems [Cyb96].

Software reuse, historically, focused on objective assets of software systems and the main practice was repackaging and reapplying of code modules, libraries or entire applications in the new software projects. Later, reuse was extended to cover more abstract entities of software systems too. It is now acknowledged as beneficial to reuse software design, aspects of project organization and methodology, development processes, and communication structures [Cyb96]. Software architecture, among the abstract entities of software systems, has been recognized as an important consideration for software reuse [FK05]. More specifically, architectural decisions and architectural patterns are regarded as facilitators of reuse in software development process. Next sections cover these concepts.

2.5 Software Architecture

Every software system has a fundamental structure known as software architecture [BCK03, TMD09]. Even when the developers intentionally have not designed the structure of the software, the software has an architecture. However, there is no unified definition of software architecture. In the software architecture community, there are at least 100 definitions for software architecture [1]. One of the most adopted definitions is from the Software Engineering Institute: ”The software architecture of a program or computing system is the structure or structures of the system, which comprise software *elements*, the externally visible *properties* of those elements, and the *relationships* among them” [BCK03]. For example in a simple airline booking website, the elements are the three layers of the software (user interface, business layer and data base layer) and the components of each layer (for example the calendar component in the user interface layer). The properties of those elements are the functional attributes of the elements (what the elements do)

[1] http://www.sei.cmu.edu/architecture/start/glossary/community.cfm

and quality attributes of the elements (how the elements do it). Finally, the relationships among the elements state how the elements interact and communicate with each other (for example the HTTP request message between two layers of the the booking website).

The focus of the mentioned definition is only on the structure of the software product. Although this definition is still being used in the software architecture community, the process that leads to the structure of the software has also become a part of the concept of software architecture. The international standard for architecture description defines the software 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” [ISO11b]. Although this definition keeps the elements, their properties and relationships as a main part of the software architecture concept, it considers the principles of design and evolution of the software as another part of the concept. These principles are mainly the design decisions that lead to the structure of the software. These decisions, known as architectural decisions, will be elaborated in the next subsection. The definition of software architecture to be used in this thesis is:

Definition - Software Architecture. Software architecture is the structure of a software system which compromise its components, the properties of those components, and the relationships among them, alongside the decisions that lead to that structure.

2.5.1 Architectural Decisions

Architectural decisions are considered a first class entity in software engineering [ISO11b]; researchers define software architecture as a set of architectural design decisions[JB05]. Architectural decisions are design decisions concerning a software system as a whole, or one or more of its components and connectors in any given view [Zim09]. They are the outcome of the architectural decision-making process where architects choose one alternative among various alternatives for solving an architectural issue. Therefore the triple of issue-alternative-outcome is the essence of the decision-making process. For instance, in the scenario of developing an airline booking website, one *architectural issue* is choosing the topology of components in the physical level. The *alternatives* are a thin client or a thick client, where in the thin client only the components of the user interface layer will be located on the client whereas in the thick client the components of user interface and business layers will be located on the client. Each of the alternatives have some pros and cons and the architects choose one alternative based on the drivers of the system (e.g., quality attributes, time-to-market, etc.). The chosen alternative is regarded as the *outcome* of the decision-making process. Other instances of architectural issues for developing a booking website would be the distribution of databases, data caching strategy, type of API (e.g., RESTful) and so on. For each of these architectural issues, the architects should identify the possible alternatives and choose one alternative based on the requirements of the system and concerns of the company.

When a software system is small and centralized, designing its architecture is not challenging. The architects can identify the architectural issues in a straightforward manner even if they only rely on their intuition and previous experiences. But when the system grows in various aspects (size, distribution, number of stakeholders, etc.), the decision-making process becomes very challenging. Different parts of the system might be developed in various development departments and commercial-off-the-shelf (COTS) or open-source components might be acquired and reused. Such components might be purchased from various vendors and therefore interoperability and integration becomes an issue due to architectural mismatch [GAO95]. Besides the technical challenges, organizational challenges also play an important role in decision-making process when the developed system is a ULSS. However, how various actors of a SECO affect the architectural decisions that each of them make, has not been focused in the literature so far.

Definition - Architectural Decision. An architectural decision is a choice to be made or has been made among several alternative options for solving an architectural issue.

Architectural Decisions vs. Architectural Patterns

Architectural patterns are reusable solutions to recurring issues in software architecture within a given domain [TMD09]. Therefore, sometimes architectural decisions are confused with architectural patterns. To make the difference between the two clear, we refer to the words by Harrison et al.:

"The major difference between architecture patterns and architectural decisions is in the scope of information each contains. Each architectural decision document describes an individual decision about the target system. In contrast, patterns describe solutions that have proven successful in multiple applications. Thus, architectural decisions are specific, but tentative; patterns are proven, but general. When designing systems, architects consider patterns as alternative solutions. ... Architectural decisions comprise application-specific knowledge, whereas architecture patterns comprise application-generic knowledge" [HAZ07].

2.5.2 Architectural Debt

Driven by development costs, time-to-market, and other stakeholder concerns, software development projects often strive for maximum business functionality as early as possible with little regard to software quality attributes. This often leads to a phenomenon that metaphorically has been called *technical debt (TD)* in the software engineering community since it was introduced in 1992 by Cunningham [Cun92]. TD refers to technical compromises that can provide short-term benefit for project stakeholders but may hurt the long-term health of a software system [LAL15]. Originally described by "not quite right

code which we postpone making it right" [Cun92], various people have used the term to describe many other kinds of ills in software development: test debt, architectural debt, requirement debt, documentation debt, etc. [KNO12]. Architectural debt (AD) is a type of TD which is caused by architecture decisions that consciously or unconsciously compromise software quality attributes [LLA15]. When short-term business advantages are the main focus, AD is not necessarily a "bad thing"[LLA14]. However, the invisible nature of AD (like other types of TD), makes it necessary to be managed sooner or later, as it is accumulated over time and may violate system quality attributes; especially maintainability and evolvability [KNO12]. Figure 2.4 shows the landscape of TD, the invisibility of TD (and AD) in this landscape, and the issues and challenges that TD may bring into the future of software development cycles (maintainability and evolvability issues).

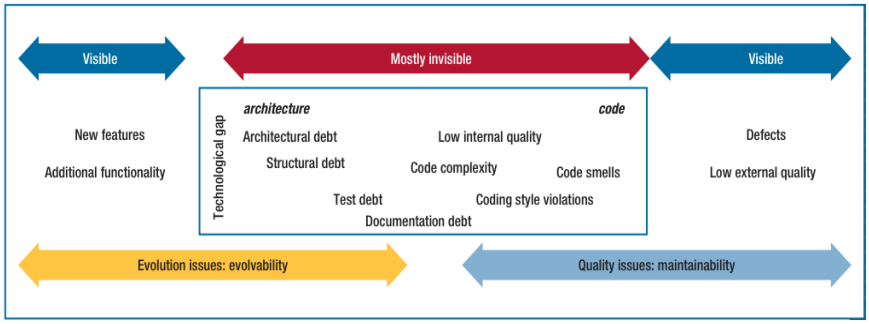


Figure 2.4: The technical debt landscape [KNO12]

Besides the architectural decisions that consciously produce AD due to short-term business related benefits, one of the sources for making unsound architectural decisions (and consequently AD) are cognitive biases that architects unconsciously encounter. Therefore, diminishing such biases may increase the quality of architectural decision-making processes and lead to lower AD in the projects. The next subsection elaborates cognitive biases in decision-making.

2.5.3 Cognitive Biases in Decision-Making

The knowledge that architects use to make decisions in designing complex systems (such as ULSS) is very diverse. Like other decision-makers, architects face cognitive biases in applying the diverse knowledge that is available to them for making decisions [RR98]. The availability heuristic and affect heuristic are two instances of cognitive biases that architects encounter.

The availability heuristic, first explained by Tversky and Kahneman [TK73], is a mental shortcut in decision making process that relies on immediate examples that come to mind and gives too much weight to recent experiences [AB14][Con14]. It occurs when several

pieces of information affect a decision, but only some of them are readily available. This often ends in decisions, based on incorrect and simplified assumptions, which need to be revised later, managed as technical debt [RR98].

Affect heuristic refers to the fact that people make decisions by consulting their feelings. In many situations, people make choices that directly express their emotions and their basic tendency to approach or avoid, in most cases without being conscious that they are doing so. People's emotional evaluations of decisions, and the approach and avoidance tendencies associated with them, all play a major role in guiding decision making processes [Kah11]. In a survey about technology selection, when people were in favor of a technology, they evaluated it as offering large advantages and imposing little risk; when they disfavored a technology, they could think only of its deficiencies, and few benefits came to mind [Kah11]. This can affect software architects in considering specific quality attributes as drivers of a specific decision and ignoring other quality attributes that might be relevant to the decision.

Generally, the cognitive biases in decision-making do not disappear just because software architects become conscious about them. Hence, it is probably not sufficient simply to make architects aware of the likelihood of the biases. Possible remedial actions are required such as framing problems to highlight relevant information that might otherwise be ignored [SM95]. Architectural decision guidance that has been proposed in the architectural knowledge research community [Zim11], highlights relevant information required for architectural decision-making in a specific project. Therefore, it can be used to reduce the negative effects of cognitive biases that architects face. As a result it helps the architects to be aware of architectural issues in the earlier phases of the projects when the cost of refactoring is much less than later phases.

2.5.4 Architectural Decision Guidance (Design Guidance)

An architectural decision guidance (or a design guidance) is a reusable asset containing knowledge about architectural decisions required when architects design a new system [Zim11]. The guidance includes various architectural knowledge entities and is enhanced based on knowledge captured from already-completed projects in a similar context or application domain, or knowledge available in literature such as guidance books and pattern catalogues. Such knowledge entities are supposed to be captured during current decision-making process or to be included in decision guidance for reuse in future decision-making processes.

Several scholars have proposed different synonyms for architectural knowledge entities. Hordijk et. al. have proposed design problem, solution option, and quality indicator as the outline of a design space [HKW04]. Tyree and Akerman have suggested issue, decision, and related requirements to be included in architecture decision description [TA05].

Jansen et. al. have proposed problem (decision topic), potential solutions, quality attributes (trade-off), and choice in their conceptual model for an architectural design decision [JBA08]. Zimmermann et. al. have proposed issue, decision drivers, alternatives, outcome, and rationale as the entities of a metamodel for architectural decision capturing and reuse [ZKL+09]. Gu et. al. have developed a template for SOA design decision making where design issue, quality attributes, architectural options, and rationale are essential elements of the template [GLVV10]. ISO/IEC/IEEE 42010 as a standard for architecture description proposes decision, rationale, and concern to be captured in a decision-making process [ISO11b].

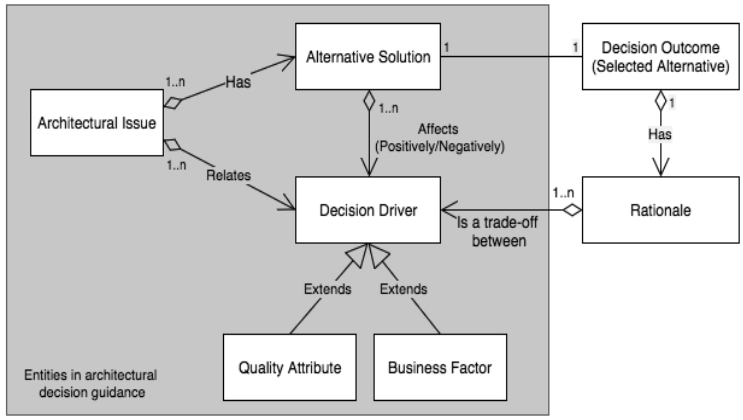


Figure 2.5: Entities/relationships in an architectural decision-making process and the scope of architectural decision guidance

Figure 2.5 shows the entities (and their relationships) that have been selected among the mentioned proposals, for describing architectural decision-making process in this thesis. An architectural decision guidance, is essentially a list of potential architectural issues that architects should consider in new projects. An architectural issue informs the architect that a particular design problem exists and requires an architectural decision. Issues relates to decision drivers, namely quality attributes and business factors, and have alternative solutions along with their advantages (pros) and disadvantages (cons)[Zim11]. As a reusable asset [Gro05], an architectural decision guidance has been curated, edited and quality-assured by a knowledge engineer for readability and reuse.

Figure 2.6 shows a snapshot from an architectural decision guidance for a web development project.

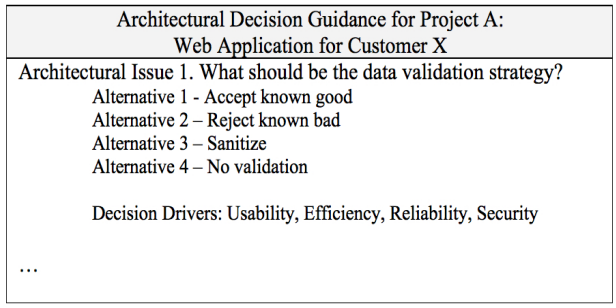


Figure 2.6: A snapshot from an architectural decision guidance (design guidance)

2.5.5 Architecture and Agility

Software architecture and *agile software development approaches* have both shown their significant impact on software engineering practices and products since emerging their arrival; both have proponents and opponents in the industry and academia. While software architecture has been used as a mean to increase the quality of the products, many in the agile community consider it as an example of the anti-pattern of *big design up front*. Empirical observations also show that practitioners in the agile projects see the architectural design problematic [PD12]; as a result the crucial role of underlying architecture in these projects are often overlooked [BNO10]. On the other hand, while agile development processes have been adopted to meet the industry demands for rapid delivery of the products, many of architecture's advocates see them as lacking sufficient forethought and rigor [Gru13]. Many practitioners in the domains of ultra large scale systems (e.g. telecommunications) doubt the scalability of agile development approaches that do not pay sufficient attention to the architecture of systems [ABK10].

The importance of architecture and agility have made the researchers in the software engineering community to bring the idea of companion and co-existence of the two into the community [ABK10]. In this regard, to build a bridge between the two domains and to use the benefits that both offers, researchers in the last five years have proposed ideas, approaches and frameworks to align software architecture and agile processes [BBM13][EKGER][HEK15][Álv13]. Prior to this thesis, several case studies have reported that applying architectural decision guidance improves speed in software design activities [Zim09]. The solution that this thesis proposes intends to encourage practitioners even more to adopt systematic architecture design in their agile development approaches through an information extraction framework that accelerates the orientation in the problem-solution space.

2.6 Information Extraction

Natural language processing (NLP) is a range of computational techniques for the automatic analysis of natural language of humans. NLP research has evolved from the era of punch cards and batch processing to the era of Google and similar solutions. Since its inception in 1950s, NLP research has been focusing on tasks such as machine translation, information retrieval, text summarization, question answering, information extraction, topic modeling, and more recently, opinion mining [CW14].

With roots in the NLP community, Information Extraction (IE) refers to the task of extracting structured information such as entities, relationships between entities, and attributes describing entities from unstructured sources, mainly text documents [Sar08]. The topic of structure extraction now engages many different research communities such as machine learning, information retrieval, databases, and web analysis [Sar08]. It has several applications in different domains. Examples are scientific applications such as extracting biological objects from text documents in the bio-informatics domain, enterprise applications such as tracking events from news sources, and web oriented applications such as sentiment analysis in opinion databases [Sar08].

As the definition explains, any IE system has unstructured information as input and structured information as output. Type, granularity and heterogeneity of input and output information affects the complexity of the extraction task and determines the architecture of the IE system. The type of the output information can be the extracted entity itself, the relationship between extracted entities, the adjective describing the entities, or structures such as ontologies [Sar08].The granularity of the extracted information can vary from word level level to sentence level, and finally to document level. Heterogeneity of the format and style of the input information is another important concern that has a huge impact on the complexity and accuracy of the IE system [Sar08]. On the more homogenized side of the heterogeneity spectrum the machine generated text are located (e.g., HTML documents dynamically generated via database backed sites). The less homogenized input information comes from partially structured domain specific sources (e.g., news articles). At more heterogenized end of the spectrum open ended sources are situated (e.g. the web where there is little that can be expected in terms of homogeneity or consistency) [Sar08].

In this thesis, the goal is to design an IE system that receives text of guide books, industrial standards, and informal documents from companies. The output is a list of sentences that contain architectural issues. Therefore, the *type* of extracted information is *entity*, the *granularity level* is *sentence*, and the input information is very *heterogeneous*.

2.6.1 Methods of Extraction

Several methods have been introduced for extraction of information. Sarawagi has categorized the extraction methods along two dimensions: hand-coded or learning-based, and rule-based or statistical [Sar08].

A hand-coded approach requires human experts to develop rules, regular expressions, or program snippets for performing the extraction task. That person should be a domain expert and possess linguistic understanding to be able to develop extraction rules. In contrast, learning-based approaches require manually labeled unstructured examples to train machine learning models of extraction. Even in the learning-based approaches, domain expertise is required in annotating examples that will be representative of the actual deployment setting. It is also necessary to have an understanding of machine learning to be able to choose between various model alternatives. The nature of the extraction task and the amount of noise in the unstructured data are the indicators for deciding between a hand-coded and a learning-based approach [Sar08].

Rule-based extraction methods are driven by hard predicates (rules), while statistical learning methods make decisions based on a weighted sum of rule firings. Rule-based methods are easier to develop, whereas statistical methods are more robust to variation in the unstructured text. Hence, rule-based methods are more useful in domains where human involvement is both necessary and available. The statistical methods, on the other hand, are more appropriate in broad-range domains like opinion mining from blogs [Sar08].

In the rule-based extraction methods, rules consist of a condition part and an action part. The condition part of the rule is a pattern of properties which need to be fulfilled by a position in the text (e.g. a word or a sentence). Such properties may be capitalization, formatting, or presence in a list of terms. If the condition is met (the pattern matches on a text position), the action part of the rule is conducted. In most of the cases, the action part is adding the text position to the list of annotations [Klü15]. For example, an extraction application that is run to identify people names in texts may have a rule as follows:

Condition - Wherever there is a ”Mr.” or ”Mrs.” followed by a noun which starts by a capital letter,
Action - Annotate the noun as a person name.

2.6.2 Metrics for Evaluating Information Extraction Systems

Recall and precision are the main metrics used in the information extraction domain to measure how well an information extraction system retrieves the relevant entities requested by a user [SW11]. The metrics are adopted to this thesis as follows:

$$Recall = \frac{\text{the number of annotated sentences that are relevant}}{\text{the number of relevant sentences in the reference document}}$$

$$Precision = \frac{\text{the number of annotated sentences that are relevant}}{\text{the total number of annotated sentences}}$$

Besides, we define a third metric to be applied in the evaluation part of the thesis: Effort Reduction measures how much effort an information extraction system reduces reading of the architectural documents; by extracting the sentences that contain architectural issues. It is calculated as:

$$Effort\ Reduction = 1 - \frac{\text{the number of annotated sentences}}{\text{the total number of sentences in the document}}$$

The number 1 in the right-hand side (RHS) of the equation represents the effort of reading the whole document manually. The portion in the RHS represents the effort of reading only sentences that an information extraction framework has extracted from the document. The effort reduction is therefore the subtraction of this portion from 1. All metrics are presented in percent. They are visualized in Figure 2.7 [2].

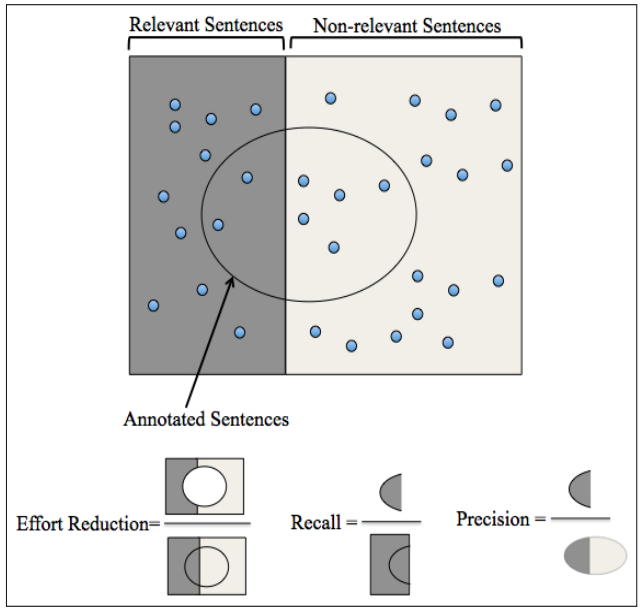


Figure 2.7: Metrics to be used in the evaluation section of thesis

[2]The figure is inspired by http://en.wikipedia.org/wiki/Precision_and_recall

2.7 Research Methods in Software Engineering

As Victor Basili stated in his essay published in 1993, engineering, empirical and mathematical methods are three of the main research paradigms that can be applied in software engineering research projects [Bas93]. The two former methods, engineering and empirical, have been used in this thesis. "The *engineering method* observe existing solutions, propose better solutions, build or develop, measure and analyze, repeat until no further improvements are possible" [Bas93]. Other synonyms of the term might be used such as *development method* [Sha02] or *design science*. The empirical method consists of a broad range of research types, data collection methods, and data analysis approaches. Therefore, it is elaborated in more details in the next subsection.

2.7.1 Empirical Methods

In software engineering research in general, without knowing the fundamental mechanisms that derive the costs and benefits of software tools and methods for a certain application, researchers can not say whether they are basing their contributions and actions on faulty assumptions. In fact, unless they understand the specific factors and drivers that cause tools and methods to be more or less cost-effective, the development and use of a particular technology will essentially be a random act. Empirical studies are a key way to get this information and move towards well-founded decisions and actions [PPV00]. It is the same situation in the architecture area. Conducting empirical studies and observations would be the base for evaluating effective methods, frameworks and tools.

Research takes many forms and it may be challenging to know which research approaches and research methods to apply in different situations. As a researcher, the different alternatives work as a toolbox, and it is far from trivial to choose the right tool in a given study [WA14]. Therefore, selecting appropriate empirical research method is a challenging decision-making task. Wohlin and Aurum have developed a conceptual research decision-making structure for selecting an appropriate research method. It is illustrated in Figure 2.8.

In the following parts, the empirical research methodologies, data collection methods, and data analysis methods that have been used to answer the research questions of this thesis, are briefly introduced. In Chapter 3 the chosen methods for answering each of the research questions are described. The detailed design of the conducted studies, the characteristics of each of the methods and the rationale for selecting the methods are presented in the papers at the Part 2 of this thesis.

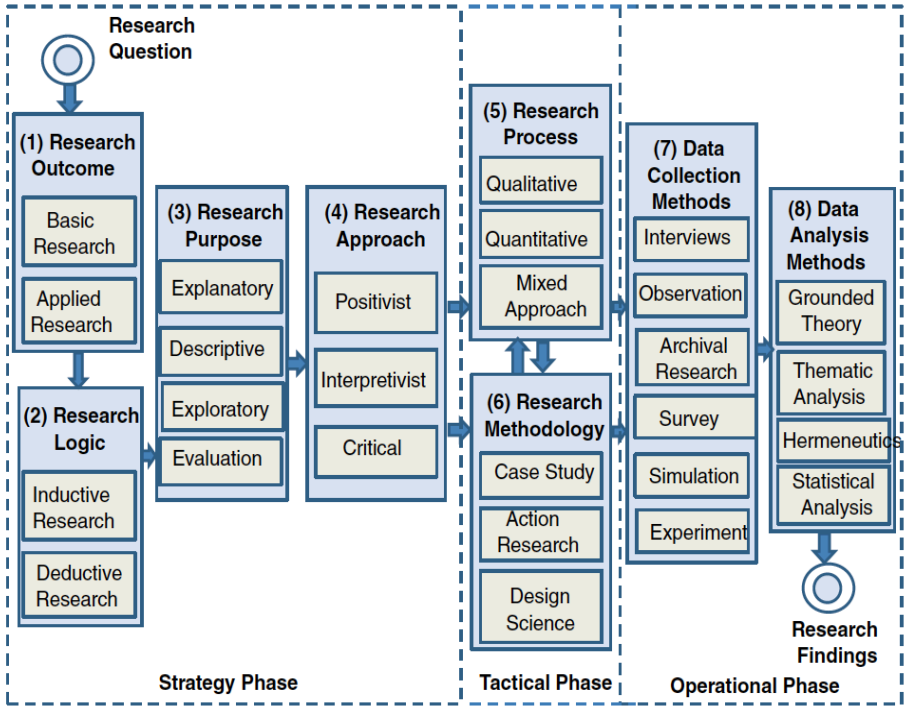


Figure 2.8: Research decision-making structure [WA14].

Qualitative Interview

"The qualitative research interview is a construction site of knowledge. An interview is literally an inter view, an interchange of views between two persons conversing about a theme of mutual interests" [Kav96]. There are three general types of research interviews: structured interview that is conducted through a structured questionnaire, semi-structured interview that is conducted on the basis of a loose structure consisting of open ended questions, and unstructured interview that is less structured and usually covers one or two research problems in great details [Bri95]. Considering the context of interview, more than one informant (group interviews or focus groups) and interviews by telephone are another types of interviews [BB11]. In qualitative research, the two main types of interviews are semi-structured and unstructured interviews. In the semi-structured interview the researcher has a list of questions on specific topics to be covered, but the informants have a great deal of freedom in responding to the questions. Questions may not follow in the way outlined on the questionnaire. Questions that are not included in the questionnaire may be asked based on the answers of informants. But all the questions will be asked and a similar wording will be used from interview to interview [BB11].

Case Study

"Case study is an approach to research that facilitates exploration of a phenomenon within its context using a variety of data sources. This ensures that the issue is not explored through one lens, but rather a variety of lenses which allows for multiple facets of the phenomenon to be revealed and understood" [BJ08]. According to Yin a case study design should be considered when [Yin14]:

(a) The goal of the study is to answer "how" and "why" questions,

(b) Researchers cannot manipulate the behavior of those involved in the study,

(c) Researchers want to cover contextual conditions because they believe the conditions are relevant to the phenomenon under study, or

(d) The boundaries between the phenomenon and context are not clear [Yin14].

Case studies are categorized into single or multiple, explanatory, exploratory or descriptive, and intrinsic, instrumental or collective [BJ08]. Depending on the goals and limitations of the study, researchers decide the type of the case study.

Experiment

In very general terms, an experiment is trying new things and seeing what happens, what the reception is. However, when experimentation is considered as a research design, it involves the control and active manipulation of variables by the experimenter. An experiment is a research methodology involving:

- "the assignment of participants to different conditions;
- "manipulation of one or more variables (called independent variables, IVs) by the experimenter;
- "the measurement of the effects of this manipulation on one or more other variables (called dependent variables, DVs); and
- "the control of all other variables" [Rob11].

A randomized controlled trial (RCT) is a type of experiment where participants are randomly allocated, either to a group who receive some form of treatment, or to a group who do not [Rob11]. There are several alternative designs for conducting an RCT. Some commonly used designs are two-group designs, three- (or more) group designs, factorial designs, parametric designs and repeated measures designs [Rob11]. Experimenters take many considerations in choosing among experimental designs, dependent on the goal and limitations of the study.

Survey

A survey is a data collection method in which respondents answer questions that were prepared in advance [Kas05]. It should be distinguished from a literature survey where researchers review and synthesize literature for understanding state-of-the-art in a specific topic. Survey can be conducted through different approaches: oral survey by telephone, paper-based survey distributed by mail or conducted face to face, and web-based survey distributed by email or through posting on social networks. In a survey, both qualitative information and quantitative data can be gathered. A survey should be conducted on a sufficient sample size to make the quantitative results generalizable. The proper sample size depends on factors such as the studied population and margin errors [KH01].

Thematic Analysis

Thematic analysis or synthesis is a method for analyzing the textual data gathered by a systematic literature review or collected by a qualitative interview. It is a stage-by-stage method and the main goal is to increase the abstraction level of transcribed texts from the text level to the code and theme level and then create taxonomy of higher-order themes[Bur91, CD11]. Cruzes and Dybå recommend four steps for thematic analysis of extracted textual data in software engineering:

1. Identify and annotate interesting concepts, categories, findings, and results as codes across the entire data set,
2. Merge and translate codes into themes, sub-themes and higher order themes,
3. Explore associations between themes and create a model of higher-order themes,
4. Assess the trustworthiness of the interpretations leading up to the thematic analysis [CD11].

Applying thematic analysis method creates categorizes of themes from several pages of unstructured texts. The final groups of themes are usually presented as the answers to the research questions of the study.

CHAPTER 3

Research Design

This chapter briefly describes the research methods used in this thesis. The definitions of the research methods were presented in Section 2.7. This chapter also describes the research evolution of this PhD.

3.1 Research Questions and Research Methods

This section describes which research methods have been employed to address each of the research questions.

RQ1: What is the attitude of large-scale enterprises in making and reusing architectural decisions and how do available tools and research prototypes support them?

A set of qualitative interviews have been conducted in the industry to explore the state-of-the-practice regarding making and reusing architectural decisions. Also, an online survey has been carried out to measure the degree of agreement among experts in architectural decision-making process. Besides, a literature review has been conducted to investigate how the current tools and research prototypes support reusing architectural decisions.

Within the exploratory study, eight qualitative interviews were conducted in six large-scale organizations in the Norwegian electricity industry. They are all actors in the software ecosystem (SECO) that delivers the required software applications for the electricity industry in Norway. Five of the organizations are grid utilities (customers of SECO) and

one was a software development company (ISV of SECO). The details of the informants, the organizations, the interview guide, and the thematic analysis used to analyze the interview transcripts, are presented in Paper 1 (see Chapter A).

The survey was conducted at the end of the PhD journey. The goal was to find out whether experts have agreement when they associate an architectural issue to relevant quality attributes in architectural decision-making process. The findings of the survey are suitable for addressing RQ1.The details of the survey is presented in Paper 5 (see Chapter A).

The literature review (tool analysis) was conducted to find the latest developed approaches and tools that assist practitioners in reusing architectural decisions across their projects. The details of the literature review and the list of the tools analyzed, are presented in Paper 2 (see Chapter A).

Conducting a set of qualitative interviews, a survey, and a literature review addressed the first research questions and led to initiating the second research question.

RQ2: How can a framework be established to develop architectural decision guidance from architecture-related documents in a rapid way?

A rule-based NLP framework is the main solution of this thesis for enhancing architectural guidance from architectural related documents in a rapid way. To find out how such a rule-based framework should be developed, the engineering method (design science) has been used. To test and improve the framework, a case study has been conducted. Based upon the results of the case study, the framework was improved. The best method of developing the framework was chosen among all the tested methods in the case study. The details of the development are available in Paper 3 (see Chapter A) and the details of the case study is presented in Paper 4 (see Chapter A).

RQ3: How efficient and effective will such a framework be in developing architectural decision guidance?

This research question addresses the evaluation of the rule-based framework developed during this research. The case study (presented in Paper 4 in Chapter A) was also designed for finding out how efficient and effective the framework is in enhancing the architectural guidance from architecture-related documents. Besides, two experiments were conducted to find out among automatic and semi-automatic information extraction approaches which one operates more efficient and therefore should be included in the framework. The first experiment involved 19 students of an IT programme while the second experiment was conducted on 21 experienced software/IT architects from five Norwegian companies that develop or integrate ultra-large-scale systems. The details of the design and participants of the first experiment are available in Paper 3 (see Chapter A) and the details of the second experiment are presented in Paper 4 (see Chapter A).

The details of each method are presented in the papers available in Part 2 of the thesis. A summary of the methods are shown in Table 3.1. It only includes the research strategies

and data collection methods. The data analysis methods and statistical tests are only presented in the papers.

Research Method	Setting	Research Question	Paper
Qualitative Interviews	Interviewing 8 informants (IT managers and architects) from 6 organizations in the Norwegian electricity industry	RQ1	Paper 1
Survey	An inter-rater agreement study on 37 IT experts	RQ1	Paper 5
Literature Review	Analyzing 5 tools and research prototypes	RQ1	Paper 2
Design Science	Developing and testing an information extraction framework	RQ2	Paper 3 and 4
Case Study	Using 3 architectural documents from a telecommunication company to test and evaluate the framework	RQ2, RQ3	Paper 4
Experiment 1	Evaluating the framework with 19 IT students	RQ3	Paper 3
Experiment 2	Evaluating the framework with 21 expert architects from 5 Norwegian organizations	RQ3	Paper 4

Table 3.1: Summary of research methods

3.2 Research Evolution

This PhD was a journey that traversed more topics than those presented in this thesis. The initial research topic was Improved Management of Software Evolution for Smart Grid Applications. In the inception phase, we made some preliminary observations in the electricity industry and in the literature. Two conceptual papers were published reporting the conducted literature reviews. One paper is still slightly connected to this thesis and is presented as a supporting paper (Paper 0) while the other one is not included in the thesis [ACC12]. The research topic after 1.5 years shifted from the focus on software evolution in Smart Grid to architectural decision making for large-scale software systems. Smart Grid as an instance of large-scale systems is still connected to the research, but the research also concerns other large-scale systems such as those in telecommunication or banking area.

Figure 3.1 shows how this PhD research has evolved over the time (it excludes the first 1.5 years of the research where the topic was different). It demonstrates different phases of the research and different studies that have been conducted in each phase and the papers that has been delivered during the research. After conducting several qualitative interviews, the more specific interest of the research became reusing architectural decisions

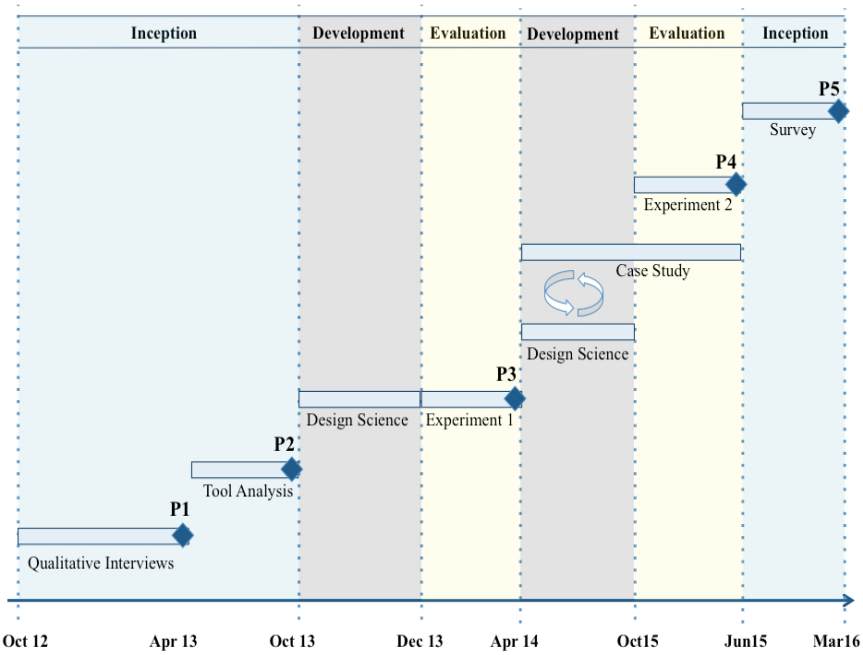


Figure 3.1: The research evolution and phases

and enhancing architectural decisions guidance from previous decisions. A literature review analyzed the current tools that support reusing architectural decisions and enhancing decision guidance. Completing the inception phase, accelerating the decision guidance enhancement became the main focus of the research. Several methods were employed to develop and evaluate a framework supporting automated decision guidance enhancement. The journey ended by conducting a survey to initiate an inception phase for the future study. It should be noted that the path shown in Figure 3.1 is a retrospective view on the journey; the PhD roadmap was not designed prospectively in this manner. It was evolved over time and the final roadmap is presented here.

CHAPTER 4

Results and Analysis

This chapter summarizes the results of this thesis. The subsections are written based on the results of studies which were introduced in Chapter 3. Each section gives a summary and contribution of the results of each step of the research. The details of the results are available in the selected papers presented in Part 2 of the thesis.

4.1 Architectural Decision-Making in Enterprises: State-of-Practice

The first step of this PhD research was to explore the large scale enterprises to find out the main processes and issues on making and reusing architectural decisions by considering the relationships among the enterprises and other actors of the ecosystem. This exploratory study partially answered the first research question of this thesis which is presented in Section 3.1. The full results of this study is published in Paper 1. In the following, the summary of the results are presented.

4.1.1 Architectural Decision-Making Approaches

The results of our study in line with the literature [vHA11], show that most of the large scale enterprises are not using systematic approaches such as the architectural trade-off

analysis method (ATAM) [KKB+98] to make and evaluate their architectural decisions. Nevertheless, it does not imply that the organizations are making their architectural decisions totally unsystematically. Both the results of our study and findings from literature, show that the enterprises first identify architecturally significant requirements (architectural analysis), then find different candidate solutions for the requirements (architectural synthesis), and finally validate the chosen solution against the requirements (architectural evaluation) [vHA11]. In spite of similarities, different companies of our study have various procedures for each of the mentioned processes. For instance, for architectural evaluation, some apply proof-of-concept, while some launch industrial prototype to evaluate the chosen solution.

4.1.2 Effect of Software Ecosystem Relationships on the Architectural Decisions

The case of our study was the software ecosystem (SECO) of the Norwegian electricity industry. Figure 4.1 visualizes the software supply network in this SECO.

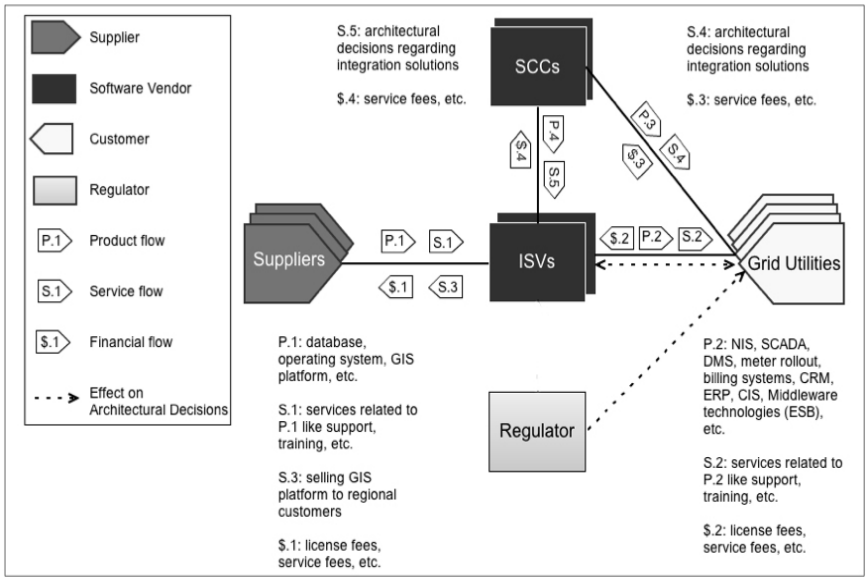


Figure 4.1: Current software supply network in the Norwegian electricity industry

The results of our study show that the relationships among the actors of a SECO could significantly affect the architectural-decision making process for each of the actors. In this study, as Figure 4.1 shows, these effects are : 1) effect of regulators on the architectural decisions of customers, 2) effect of vendors on the architectural decisions of customers, and 3) effect of customers on the architectural decisions of vendors.

4.1.3 Reusing Architectural Decisions across Projects

One of the aims was to find out whether the explored enterprises identify the required architectural decisions in a new project based on experiences from previous projects. The results show that some of the enterprises reuse high-level architectural decisions in term of architectural guidelines and rules set by their IT department. However, when it comes to the low level architectural decisions, almost none of the explored enterprises are reusing the decisions across different projects. Although some of the enterprises document important architectural decisions, the decisions are not transferred between different projects in a written sense. Therefore, some of the organizations show interest to become familiar with reusable architectural decision frameworks. They believe that it would be useful to learn from history and apply the experiences from previous projects in future projects.

Contribution. The first step of the PhD identified the general attitude of large-scale enterprises in making significant architectural decisions and reusing such decisions across various projects and departments. Besides, it pictured the possible effect of relationships among various actors in a software ecosystem on the process of making and reusing architectural decisions at each actor.

4.2 Reusing Architectural Decisions as Design Guides: Tool Analysis

The observations from the large scale enterprises (presented in Section 4.1), show that there is both a gap and an interest in reusing architectural decisions from previous projects in new projects. Reusing architectural decisions from previous projects as design guides for new projects gives these decisions a more proactive role and therefore makes decision management more appealing to practitioners. Although in new projects practitioners may make different (even contradictory) decisions compared to previous projects, architectural issues (the topic of previously made decisions), possible alternatives and the decision drivers are still the highly informative knowledge to be transferred from previous projects to new projects.

In the second step of the PhD research, we leveraged our industrial experiences, our previous research work and also the current literature in the architectural knowledge community to establish functional requirements for future knowledge management tools that enhance architectural decisions to design guides. With respect to the functional requirements, we analyzed representative tools and research prototypes. The full results of the analysis is published in Paper 2 (P2). P2 reported that the available tools and research prototypes have made significant contributions in the area of architectural knowledge capture, but still require a number of extensions so that the captured decisions can serve as design guides in practice. We finalized the paper with a vision for method integration and tool

improvement. The literature exploration complements the results of the previous study in answering the first research question of this thesis.

Contribution. The second step of the PhD specified the requirements for tools that facilitate the post-processing of architectural knowledge from projects and enhancing such raw knowledge into design guides for future decision making activities. Furthermore, it analyzed existing tools and research prototypes with respect to the proposed requirements.

4.3 Semi-automated Design Guidance Enhancer (SADGE)

Chapter 2 explained that architectural decision guidance (design guidance) has been proposed in the software architecture community as a mean for improving the quality and speed of architectural decision making processes. However, the manual effort and time that practitioners should spend to enhance architectural decision guidance out of their project documentations have been a barrier for integrating architectural decision guidance into decision-making processes. An architectural decision guidance would increase the agility of the project, whereas enhancing the guidance would decrease the agility. To diminish this contradiction, a rule-based framework was developed in the third step of this PhD. The framework accelerates developing architectural decision guidance by applying techniques from the information extraction field. The framework, that is the main contribution of this thesis, extracts architectural issues from architecture-related documents available in the companies and in the literature. As it was discussed in Section 2.5.4, architectural issues are the essence of architectural decision guidance, and assist architects to reduce the effects of cognitive biases in their architectural decision-making processes.

The framework includes both automatic and manual processes, and therefore it is considered a semi-automated framework. We have used the synonym of architectural decision guidance, design guidance, in the name of the framework to make the abbreviation of the framework more readable: Semi-automated Design Guidance Enhancer (SADGE). SADGE is the result of conducting the engineering method: it was developed in the third step of the PhD, and evaluated in the fourth step in an iterative rather than a sequential manner. This section covers the result of the development step, while Section 4.4 presents the summarized results of the evaluation step. The detailed results of both development and evaluation steps are available in Paper 3 and Paper 4 in Part 2.

A motivating scenario is presented in the following to make the application of SADGE in architectural decision-making process objective and clear. The components of the framework and the workflow of its operation are described afterwards.

4.3.1 Motivating Scenario

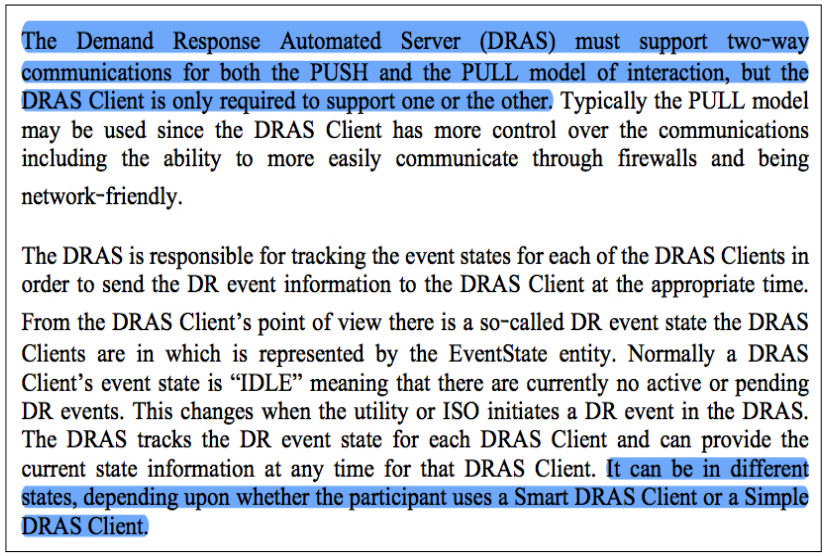
Organization A1 is a large grid utility with more than 100.000 customers. Mandated by the national organization for energy regulation, A1 should automate all optimization between electric supply and demand which requires employing new software systems. A1 is outsourcing the development of software systems to Company A2 which is an independent software vendor (ISV) in the electricity industry. Software architect M is involved in the architectural decision making activity within the development of client-server system that automates the demand-response optimization in A1. Company A2 has done a similar project for another customer before; however, architect M was not involved in the previous project. Some of the decisions from the previous project were informally documented (i.e., in documents without a predefined structured template). These documents include mailing lists, meeting minutes, and a company-internal wiki. The decisions blend in with other information in the documents, and need to be enhanced into an architectural decision guidance to be consumable for the new project. A new regulation from the national energy regulator mandates the use of the Open Automated Demand Response Communications Specification (OpenADR), which have been published on the Web[1]. As a requirement from A1, architect M is also supposed to consider this specification when making and justifying the architectural decisions in the new project.

For enhancing the architectural decision guidance for the project, M intends to highlight and identify the list of the architectural issues that (s)he should make decisions about. M will avoid identifying the required architectural issues intuitively (just based on his/her experience). M is going to apply a manual decision identification technique [Zim09] to extract architectural issues (decisions required) from documents of previous projects (wiki, mailing lists, meeting minutes), technical reports, and standards including the OpenADR specification. When the list of architectural issues is ready, M will find available alternatives to each issue (as well as advantages and disadvantages of these) to generate the architectural decision guidance for the project. Architect M will be more confident about the decision making process by having this guidance.

Figure 4.2 contains a snapshot from the text of the OpenADR specification. M wants to extract some of the architectural issues from this document. The document is a 120-page text, and M would spent 12 hours to manually annotate the sentences that contain architectural issues (these numbers are calculated based on our own annotation of one chapter of the document, extrapolated to the scale of the whole document). Figure 4.2 shows a portion of the document and two of the sentences that M has annotated that contain architectural issues. Architectural issues (decisions required) in these sentences are: *Model of interaction that DRAS client should support* and *type of DRAS client to be developed for customers*. 12 hours for covering one of the documents show that the manual annotation process is very time-consuming. If M needs to manually identify issues in all relevant documents (i.e., documents related to the previous project and the industrial

[1]Available at: http://openadr.lbl.gov/pdf/cec-500-2009-063.pdf

guidelines), M would have to spend several weeks. This would cost precious time and increase costs considerably.



The Demand Response Automated Server (DRAS) must support two-way communications for both the PUSH and the PULL model of interaction, but the DRAS Client is only required to support one or the other. Typically the PULL model may be used since the DRAS Client has more control over the communications including the ability to more easily communicate through firewalls and being network-friendly.

The DRAS is responsible for tracking the event states for each of the DRAS Clients in order to send the DR event information to the DRAS Client at the appropriate time. From the DRAS Client's point of view there is a so-called DR event state the DRAS Clients are in which is represented by the EventState entity. Normally a DRAS Client's event state is "IDLE" meaning that there are currently no active or pending DR events. This changes when the utility or ISO initiates a DR event in the DRAS. The DRAS tracks the DR event state for each DRAS Client and can provide the current state information at any time for that DRAS Client. It can be in different states, depending upon whether the participant uses a Smart DRAS Client or a Simple DRAS Client.

Figure 4.2: An example of an architectural related text and annotated sentences that include architectural issue

SADGE is developed to support company X (and similar companies) in enhancing architectural decision guidances in a more efficient and effective way.

4.3.2 SADGE Components

Automatic Annotator

Automatic Annotator (AA) is a natural language processing (NLP) tool that receives text documents and annotates sentences in them. ANNIE (A Nearly-New Information Extraction System) is used as AA in SADGE. ANNIE is a plug-in for the open source tool GATE (General Architecture for Text Engineering) [Cun02]. ANNIE has several components that run processes that are prerequisite for applying extraction rules on a text. *Tokeniser* is one of the components and splits the text into simple tokens such as numbers, punctuation and words. Another module is *sentence splitter* which segments the text into sentences. *Gazetteer* is another component which identifies entity names in the text based on lists of terms [Cun02]. By using ANNIE components, there is no need to develop basic NLP processes (such as sentence splitter) in SADGE. The customized parts required to be developed for SADGE are annotation rule and list of terms. These two are defined as follows.

Annotation Rule

Information extraction systems such as AA, need domain specific annotation rules for annotating sentences in text documents. The sentence that matches a defined linguistic pattern is extracted by the guidance of the annotation rules [KM05]. These rules are inspired by rules used by humans to process natural language texts (e.g., if a word is made up of capital letters, annotate it as a company name). Different tools formalize the rules in various ways. ANNIE uses JAPE (Java Annotation Patterns Engine), GATE's built-in language for formalizing the rules.

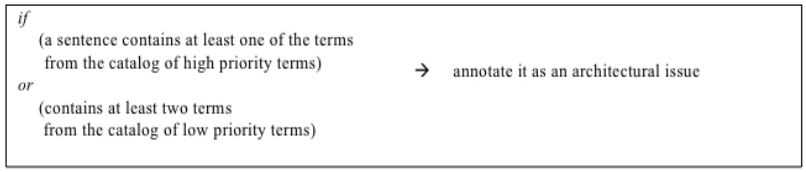


Figure 4.3: Annotation rules in SADGE

The pseudo-code of the annotation rule developed for SADGE is shown in Figure 4.3. The formal version of the rule written in JAPE is available in Appendix C. According to this rule, AA should look in a list of predetermined terms to check whether a sentence in the text contains one (or more) keyword(s) from the list. Some of these terms have higher indication value for determining a sentence as an architectural issues and forms the high priority list. The rest of the terms constitute the low priority list. A sentence should contain at least two of them to be qualified as an architectural issue. More details about the development of the Annotation Rule of SADGE, and the rationale for assigning a term low or high priority are available in Paper 3 and Paper 4.

Catalogue of Terms

The list of predefined terms that AA should look in is called the Catalogue of Terms, abbreviated CoT. SADGE has a default version of CoT. As we will elaborate later, this version should be extended by adding ad-hoc terms for extracting architectural issues from each specific document. The default version of CoT is presented in Figure 4.4. The terms are divided into different categories to make it easier to maintain the catalogue further.

Guidance Generator

The Guidance Generator is a component of SADGE that assists the knowledge engineer (e.g. Architect M in motivating scenario presented in Section 4.3.1) to categorize the architectural issues, remove redundancy, and generate the design guidance.

Low Priority Terms			
General Terms			
Verb		Noun	Adjective
apply	investigate	case	different
approach	launch	choice	several
articulate	look into	concern	various
ascertain	make	definition	
assess	measure	element	
build	pick out	establishment	
construct	plan	factor	
deal	provide	investigation	
define	recommend	limitation	
delimit	require	philosophy	
design	select	principle	
determine	set up	requirement	
discuss	specify	restriction	
employ	supply	strategy	
establish	support	type	
evaluate	take		
formulate	use		
facilitate	utilize		
find			
Architectural Terms			
Verb	Noun		
exchange	architecture	mapping	schema
refactor	class	model	service
	component	profile	topology
	framework	protocol	transaction management
High Priority Terms			
agree on	choose	decide	

Figure 4.4: Catalogue of Terms (default version)

4.3.3 SADGE Workflow

Figure 4.5 shows the four processing steps of the SADGE framework. Each step O1 to O4 are described below.

O1. Prepare a document for annotation

Input: This step takes text files as the input. The text file can be either a project document or public literature about software architecture in general or from the technical domain the project is concerned with. Project documents are often either architecture description documents or document containing architectural decisions (e.g. meeting minutes, internal wikis, etc.).

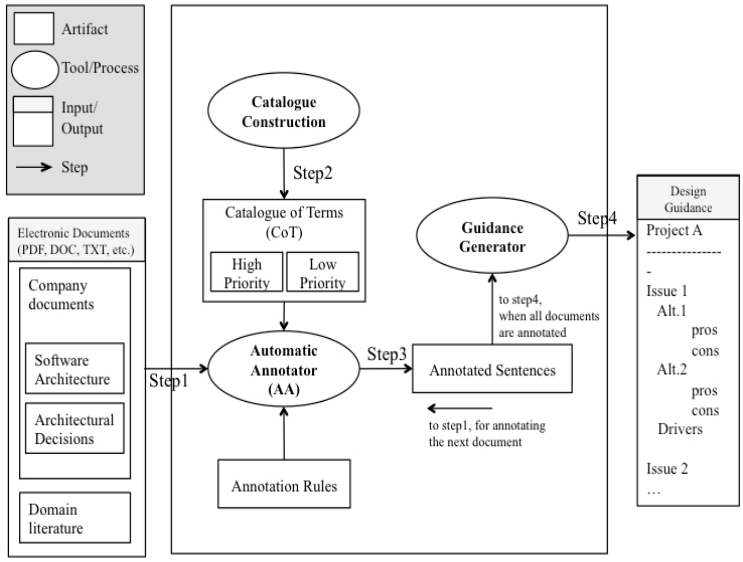


Figure 4.5: Operational stages (processing steps) in SADGE

Task: The knowledge engineer edits the text files by removing irrelevant texts (i.e., cover page, table of contents, etc.). Then, (s)he imports the text files into GATE. GATE removes any non-text objects (e.g., embedded images) automatically.

Output: This step produces batch (corpus) of pure text files.

O2. Construct Catalogue of Terms (CoT)

The knowledge engineer prepares the CoT in this step. CoT is divided into two catalogues of high priority terms and low priority terms, based on the SADGE Annotation Rule (presented in Section 4.3.2). As shown in Section 4.3.2, the high priority catalogue contains only three terms (agree on, choose, and decide) and these are universal for annotating any document. Whereas the low priority catalogue is an ad-hoc catalogue to be developed based on the input document. The knowledge engineer should construct this catalogue in a hybrid method combining an ad-hoc manual bootstrapping method with a lexical database assistance approach.

In the hybrid construction method as shown in Figure 4.6, the knowledge engineer first looks at the first 10 percent of the pre-processed text file (output of O1). The text in this portion of the document reveals the language style of the document and may introduce new clue terms that are not in the default version of CoT. In the next step (s)he uses WordNet[2] to find relevant synonyms of the new terms. New terms and their relevant

[2]WordNet is a public lexical database for the English language [Mil95]. Using lexical databases such as WordNet to look for synonyms of a term, is common in the information extraction field [GVCC98].

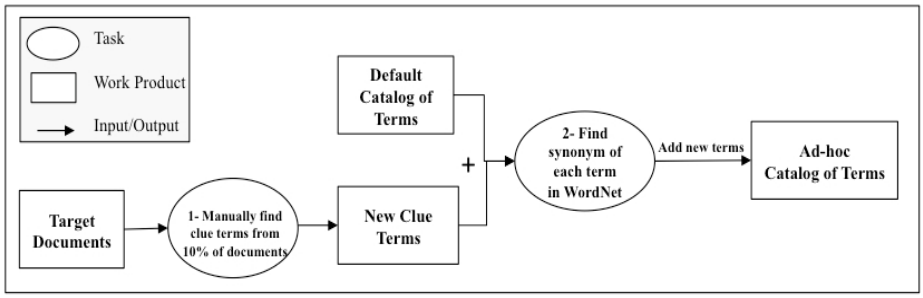


Figure 4.6: Construction of low priority catalogue

synonyms are added to the default CoT. The result is the ad-hoc CoT.

The summary of step O2 is as follows:

Input: The high-priority CoT that is fixed and universal, is the input of this step.

Task: Constructing a low-priority CoT based on each input document.

Output: This step generates the complete CoT (high priority and low priority).

O3. Automatically annotate the documents

Input: Batch of text files (output of Step1), CoT (output of Step2), and the Annotation Rules are the input of this step.

Task: AA applies the rules and annotates the architectural issues in the text file.

Output: A list of sentences that AA suggests as architectural issues is the output of this step.

O4. Generate the design guidance

Input: A list of sentences including architectural issues from all input text files is the input of this step.

Task: Since the CoT is constructed in an ad-hoc manner based on a portion of each document, steps O1 to O3 should be iterated on all documents. When the annotated sentences from all of the text files are finalized (Step2 and Step3 executed on all input documents), the Guidance Generator merges them and produces a design guidance for the project. It includes all the potential architectural issues in the project. The knowledge engineer can shorten the sentences, classify issues into groups, and add alternatives (including pros and cons for each alternative) to each issue. (S)he can also remove redundant issues. AA just extracts knowledge; it does not guarantee the correctness of the gathered knowledge. The knowledge engineer is encouraged to edit extracted issues or add her (his) own issues.

Output: The design guidance for the project is the output of this step and the final output

of SADGE.

Contribution. The third step of the PhD delivered a framework for obtaining architectural decision guidance from architectural knowledge in project documents and the domain literature. The Catalogue of Terms (CoT) as the essence of the framework, and the method of constructing CoT are two other contributions of this step.

4.4 Evaluation of SADGE Framework

The fourth step of this PhD research was to evaluate the SADGE framework. We ran two experiments and one case study for evaluating the framework. The first version of the framework included a manual post-processing step after step O3 (automatic annotation step), where a knowledge engineer would accept/reject the sentences that AA annotated. The intention of the first experiments (conducted on IT students) was to compare three approaches to extract architectural issues from documents: 1) the manual approach (without using the framework), 2) automatic approach (using the framework containing only the automatic step), and 3) semi-automatic approach (using the framework containing the automatic step and manual post-processing step). We executed the second round of the development iteration after the experiment with students, and improved the operation of SADGE framework by conducting a case study. Then, we ran the second experiment, this time with expert architects, to evaluate the improved version of the framework. We had a discussion with architects to preliminary investigate experts' opinion about the framework at the end of the experiment.

Two metrics were used in the experiments for comparing manual, automatic and semi-automatic approaches: processing time and recall. To calculate the recall a reference document is needed. The reference documents (of two experiment groups) were constructed based on the opinion of the only one of researchers involved in the first experiment. This way of developing a reference document would cause a threat to the validity of the experiment results. Thus, in the second experiment, the reference documents were developed based on opinions of three experts in the software architecture domain through several sessions of discussions. More details about the rationale for selecting the metrics and also the stages of developing the reference document is available in Paper 4 (P4).

4.4.1 The More Efficient and Effective Approach for Extracting Architectural Issues

We conducted two experiments to find out the more efficient approach for extracting architectural issues among manual, automatic and semi-automatic approaches; the results

would indicate the nominated approach, among automatic and semi-automatic to be included in SADGE. Besides, the experiment results would evaluate the efficiency and effectiveness of the automatic approach against the completely manual approach. In both experiments (with IT students and expert architects), the participants were, in the first stage, asked to manually annotate a sample text from architectural documents. The outcome of this step is representative for the manual approach. Prior to the experiment sessions, the sample text was annotated by the Automatic Annotator (AA) to be representative for the automatic approach. In the second stage of both experiments, the annotated sentences by AA were given to the participants for accepting/rejecting AA annotations. The outcome of this step was representative for the semi-automatic approach. Details about the preparation, sessions, materials, participants and stages of experiments are published in Paper 3 (P3) and Paper 4 (P4).

Table 4.1 shows the results of the experiment with 19 IT students. Lower processing time of the automatic approach compared to the manual and semi-automatic approaches is not a surprising result. But when it comes to the recall, we did not expect that the automatic approach gives a higher result. The lower recall in the semi-automatic approach compared to the automatic approach, reveals that the participants have rejected some of the true positive results of the automatic part (AA annotations). We speculated the expertise level of the participants as the reason, as it is reported in Paper 3.

Approach	**Time (min)**	**Recall (%)**
Manual	9	38
Automatic	**0.03**	**86**
Semi-automatic	3	55

Table 4.1: Results of experiment on IT students

We conducted a new experiment to test this hypothesis, after improving the framework. This time the participants were 21 expert software/IT architects and the material were different from the first experiment. Table 4.2 shows the results.

Approach	**Time (min)**	**Recall (%)**
Manual	14	34
Automatic	**0.03**	**53.5**
Semi-automatic	7	32

Table 4.2: Results of experiment on expert architects

Although the numbers are different from the previous experiment, the same pattern has occurred again: Besides processing time that is much lower in the automatic approach compared to the other two approaches, recall is shown to be higher in the automatic approach. However, a statistic test needs to be run to find out whether this difference of recall is significant or not. To test data normality, the data was analyzed applying

Shapiro-Wilks test. Since the data was shown to have a normal distribution, a two-sample Kolmogorov-Smirnov (K-S) test was applied to compare the recall in the automatic, manual, and semi-automatic approaches.

Comparison	p-value
Automatic vs. Manual	0.001774
Automatic vs. Semi-automatic	0.0001581

Table 4.3: Results of K-S test

As Table 4.3 shows, the recall of the automatic approach is significantly higher than the manual and semi-automatic approaches, setting p-value threshold at α=0.01. A significantly lower recall of semi-the automatic approach compared to the automatic approach in the experiment with expert architects rejects the assumption about the expertise level being the reason for rejecting some of the correct suggestion of AA by participants.

In the search for justification of the lower recall of semi-automatic approach, we found that researchers in the field of psychology of decision-making have an empirical explanation: "Several studies have shown that human decision makers are inferior to a prediction formula even when they are given the score suggested by the formula. They feel that they can overrule the formula because they have additional information about the case, but they are wrong more often than not" [Kah11]. Therefore, the research in the psychology field suggests "to maximize predictive accuracy, final decisions should be left to formulas, especially in low-validity environments" [Kah11]. Low-validity environments are the domains that involve a substantial degree of uncertainty and unpredictability [Kah11]. The task of annotating architectural issues in a document has a significant degree of uncertainty. Hence, it can be considered as a low-validity context, and the lower recall of the semi-automatic approach compared to the automatic approach, is not a surprising result. We should note that final decision here is not the architectural decision, but the decision about annotating or not annotating a sentence as an architectural issue, to avoid any misunderstanding. SADGE is not an expert system replacing humans with artificial intelligence for decision-making. Rather, as a support (recommender) system it highlights recurring issues, thus empowering architects in making more comprehensive decisions.

In summary, considering the metrics of evaluation, processing time and recall, the automatic approach has shown to be more efficient than the semi-automatic approach for extracting architectural issues. In the initial version of the framework presented in Paper 3, we considered the manual fine-tune stage as a part of the framework. After the experiment with the experts, we conclude that this stage can be excluded. However, based on the results of the case study presented in Paper 4, we selected the hybrid method as the proper method for construction of the CoT. The hybrid method includes the manual bootstrapping and therefore the framework can be considered semi-automated, and its name Semi-Automated Design Guidance Enhancer (SADGE) is still justified.

4.4.2 Efficiency and Effectiveness of SADGE in Extracting Architectural Issues from Project Documents

The Catalogue of Terms (CoT) is the essential component of SADGE, as the annotation rule of SADGE attests. Therefore, the efficiency and effectiveness of the framework directly depends on it. The first version of CoT was developed by training CoT on seven sample documents we found in the software architecture literature (i.e., two industrial standards for software integration, three software design guidelines and two academic papers). Hence, SADGE may not perform with the same efficiency and effectiveness on new documents, especially on documents from companies where the writing language is more ad-hoc. We tested SADGE on three architectural description documents from a telecommunication company in Norway and on average, the recall declined 25% compared to the seven sample documents from the literature.

Three alternative methods for constructing CoT was proposed to overcome this limitation. The hybrid construction method (shown in Figure 4.6) was nominated as the most efficient method among them. The proposed methods and the results of the evaluation of their efficiency are presented in Paper 4 (P4). In this section, the efficiency of SADGE operating with the hybrid construction method is presented.

SADGE was applied on three architecture-related documents from a Norwegian telecommunication company, to measure its efficiency and effectiveness in extracting architectural issues from project documents. The original language of the documents was English. Other characteristics of the documents are presented in Table 4.4.

Document	Project	Number of Pages	Authors ID
Doc 1	4G	44	A,B,C
Doc 2	Network Infrastructure	25	D
Doc 3	Broadband	29	D

Table 4.4: Characteristics of documents used in case study

One of the researchers spent 18 hours to manually annotate and double check the sentences that contain an architectural issue in the evaluation documents. The annotated documents were used as reference to evaluate the SADGE efficiency. The results of the evaluation are shown in Table 4.5.

Considering SD, the mean of effort reduction and recall are more reliable than precision. While 78 percent recall for extracting architectural issue is significant[3], we were skeptical about the significance of effort reduction which is 60 percent. It means that the knowledge

[3]In literature, there is no baseline for a recall of extracting architectural issues to which the recall of SADGE be compared. However, one way to estimate an acceptable range for recall of SADGE is exploring rule-based systems for sentence extraction in other domains. Subjectivity classifiers in opinion mining domain is an instance of such systems [RWP05]

Document	Effort Reduction (%)	Recall (%)	Precision (%)
Doc 1	66	89	10
Doc 2	51	72	28
Doc 3	63	73	14
Mean	**60**	**78**	**17.33**
SD	7.93	9.53	9.45

Table 4.5: SADGE efficiency on project documents - results of case study

engineer should still read 40 percent of the document for enhancing a decision guidance out of the document. Our assumption was that practitioners might be reluctant to use SADGE with this effort reducation. If that would be the case, we should employ one of the other proposed construction methods where the effort reduction is higher, even though the recall is lower. We asked the participants a question at the end of the experiment with expert architects: How much time should SADGE save to encourage practitioners to use it? As Figure 4.7 shows, 70% of practitioners were eager to use SADGE if it reduces the extraction effort by at most 60%. As the case study shows, 60% is the effort reduction of the hybrid method. Hence, the hybrid method has potential in practice to be selected for construction of the CoT before the other three methods.

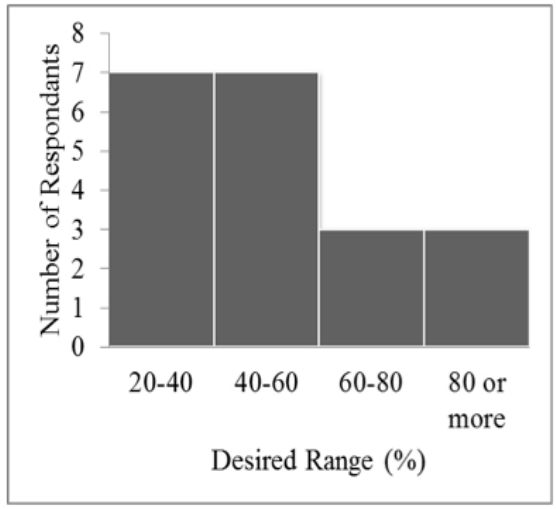


Figure 4.7: How much time should SADGE save to encourage practitioners to use it?

A question is the size of the portion of the target document that the hybrid methods are trained on (referring to Section 2.2, all of the following discussions are about the size of prepared texts that are output of Stage1; they do not include non-text objects such as figures). Is our suggestion, the first 10%, enough, too little, or too much? 10% worked well for the evaluation cases of this study that were 44, 25 and 29 pages texts. If a

document has more than 100 pages, then 10% would be too much. In that case, we suggest that practitioners only read the first ten pages of the text because we assume that ten pages of a text is enough to understand the common vocabularies used in that text since even three pages worked well in the case study. However, we have to repeat the study on documents of bigger size to test our assumption and find a more reliable answer.

4.4.3 Experts' Opinions About Usefulness and Application of SADGE

We had an oral discussion with the participants, after conducting the experiment on expert architects. The general feedback of the participants about the usefulness of the framework was positive. Some of them would read many documents in their daily tasks, and they believe that the framework can be very helpful for them to either reduce the amount of the text to read, or to determine the hot spots of their documents they need to pay extra attention to. Some of the participants also found the framework very useful for the early phases of their projects where they would find possible architectural related risks and decide about proper mitigation solutions ahead.

Contribution. The fourth step of the PhD evaluated the rule-based framework for enhancing architectural decision guidance by expert architects and IT students. It compared the automatic extraction of architectural issues to semi-automatic and manual extraction. Besides, it evaluated the efficiency of the framework in extracting architectural issues from project documents. Also, it identified the general feedback of expert architects about the usefulness and application of automatic decision guidance enhancement in their projects.

4.5 Expert Agreement on Associating Architectural Issues with Quality Attributes

The Automatic Annotator in SADGE identifies architectural issues from the input documents in an automated (rule-based) manner. Then, it is the task of the knowledge engineer to manually add alternatives (including pros and cons for each alternative) to each issue, in the last step of processing steps of SADGE. Pros and cons of each alternative are defined based on the decision drivers in the decision-making process, namely business drivers (i.e. cost, time-to-market, etc.) and software quality attributes (i.e. security, reliability, etc.). Therefore, the knowledge engineer should identify relevant quality attributes for each architectural issue to make her(him) able to position each alternative related to the identified quality attributes.

Miguel et. al. had reviewed available quality models in software engineering and identified 48 possible quality attributes (characteristics) for a software system [MMR14]. Associating an architectural issue with relevant quality attributes among 48 attributes is not

a trivial task. Conducting a survey to examines whether there is agreement or consistency among experts on associating an architectural issue with relevant quality attributes was the last step of this PhD,. If the results supported the hypothesis that there is poor agreement among experts, it would imply that task of associating is very subjective. Thus, systematic, rather than intuitive, associating each architectural issue with relevant quality attributes might be more sought in academia and more emphasized in industry.

The survey received 37 valid responses from various IT experts around the world including software/IT/enterprise architects, developers and managers. The respondents answered to five scenarios in which an architectural issues was given along seven quality attributes. The task was to give relevance score between each architectural issue and each quality attributes (0=irrelevant, 7=most relevant). Two methods were applied to measure the inter-rater agreement/reliability: Intra class correlation (ICC) and Krippendorff's alpha (Kalpha). While ICC measures observed and expected agreement, Kalpha measures observed and expected disagreement. The value of both coefficients can range from 0 to 1. When raters agree perfectly, the value of ICC and Kalpha becomes 1, which indicates perfect reliability. When raters agree as if chance had produced the results, the value becomes 0, which indicates the absence of reliability [Bar66, Kri07].

ICC became 0.12 and Kalpha 0.09 when they applied on all scenarios. The methods were also applied on each scenario and each quality attribute separately to find out whether there are specific scenarios or quality attributes on which experts have agreement. The results (presented in Paper 5 at Chapter A) show poor agreement in those categories too.

CHAPTER 5

Discussion

This chapter discusses the contributions of this PhD against the state-of-the-art. It also presents the potential threats to validity of the findings in this research. Besides, it argues the potential impact of this research on architectural decision making activities in the industry. Finally, it mentions the strengths and weaknesses of the solution this research proposes.

5.1 Discussion of Contributions Related to State-of-the-Art

This section discusses how each of the research contributions are related to the state-of-the-art in the software architecture domain.

Contribution 1-1: Identifying the attitude of large-scale enterprises in making and reusing architectural decisions and the effect of SECO relationships on their decisions

Until 2005, there were little empirical evidence about architectural decisions and how practitioners treat them in practice [TBGH05]. The motivations and insights for developing frameworks, techniques, and tools that support the architectural decision-making process in the industry have been collected mostly from researchers' personal experiences rather than empirical observation. Nevertheless, recently some empirical studies

have been conducted in this area that are discussed below.

Tang et al. conducted a survey on the use and documentation of architecture design rationale in 2005 [TBGH05]. Their main focus was to understand how practitioners think about decision rationale, how they use and document them, and what factors prevent them from documenting decision rationale. Making the decisions were not the focus of their investigations. Hoorn et al. were interested in the same direction and conducted a survey to better understand what architects really do and what kind of support they need for sharing architectural knowledge [HFLVV11].

Ivanovic and America has conducted a study to gain knowledge on information needed for architecture decisions made by architects and managers [IA10]. The reuse aspect of the decisions is not in their work. Also they have conducted their study only in one industrial organization and therefore considering the ecosystem relationships that we are interested in is not in their research.

van Heesch and Avgeriou have in their study investigated how experienced architects reason in the context of industrial projects, how they prioritize the problem space, how they propose solutions for the problem and how they choose among solutions [vHA11]. Their work is relevant to our research but still lacks the reuse aspect and does not consider software ecosystem relationships.

There is very little work in the domain of software architecture investigating agreement among experts on associating architectural issues with quality attributes. Tofan et. al. [TGL+16] have recently proposed a process, called GADGET, for increasing agreement in group architectural decision-making. They have studied the level of agreement among experts in group decision-making to show that there is a practical need for GADGET, as their first research step. The task given to the participants involves associating architectural decision topics (what we call architectural issues) with decision concerns (what we call decision drivers and include quality attributes). The researchers have studied whether conflicting perspectives occur in group architectural decision making, through a qualitative study. The participants, students and experts organized in groups of 3 or 4 members, have often reported conflicts among them when the number of decision makers has increased. This is in line with the poor level of agreement among experts reported by our quantitative study. However, our quantitative findings are gathered from larger number of experts participating from all over the world, and therefore complement the in-depth qualitative observations of GADGET in term of reliability and external validity.

Svahnberg [Sva04] has reported an industrial study where a company faces the task of identifying which among a set of architecture candidates that have the most potential for fulfilling the quality requirements of a system to build. His quantitative results shows poor agreement among 13 participants in the ranking of quality attributes within each architecture candidate and in the ranking of architecture candidates for fulfilling each quality attribute. Although we studied the agreement on associating quality attributes with architectural issues, which is more general than architecture candidates for building

a system, our study can be considered a replication of Svahnberg's work on larger number of participants.

Contribution 1-2: Perspective of existing tools and research prototypes that facilitate the post-processing of architectural knowledge from projects and enhancing decision guidance

Paper 2 is not the first survey in the software architecture domain for analyzing and evaluating the architectural knowledge management tools. At least four preceding research papers have been published [TAJ+10, Bie10, SLK09, LA09]. Although some papers have considered knowledge-sharing as a functional requirement in their evaluation framework, their focus is not on knowledge post-processing and enhancing the knowledge into architectural decision guidance, which has been the focus in our work. Furthermore, the last survey was published in 2010, while in the last years more tools and research prototypes have been developed or are under development. Paper 2 covers these new tools as well. Anyhow, the mentioned research papers were valuable for our work; for instance, we reused some of their functional requirements to establish the functional requirements presented in Paper 2.

Contribution 2: A framework for enhancing architectural decision guidance from architectural knowledge in project documents and domain literature

Sentence extraction has been used for automated requirement elicitation in the software engineering field [MBM13]. While the nature and subjectivity of a sentence that contains a software architectural issue is different from a sentence that includes a software requirement, identifying any of them in a text document may follow some common logic. Automated requirement elicitation has already been solved in the requirement engineering domain, mainly based on machine learning [MBM13, CGC10]. Therefore, machine learning can be recognized as the first option for solving architectural issue identification. However, employing machine learning to solve the architectural issue identification has some limitations:

Learning systems must be trained on large text collections before they can operate on broad and comprehensive documents in a domain [RW03]. There are large collections of publicly available online documents (e.g. open source requirement specifications documents) for training requirement elicitation methods. Whereas for training architectural issue identification methods, researchers have limited access to documents containing architectural issues. Such documents are either in-house and not accessible due to confidentiality concerns, or if available, many are not written in English. Open source projects rarely document architectural decisions. Due to these limitations, machine learning methods cannot be applied to architectural issue identification as they are applied to automated requirement elicitation. Alternatively, a rule-based NLP approach is required for enhancing architectural guidance from architecture-related documents.

Figueiredo et al. tackle similar research problems as done in SADGE; they have devel-

oped a rule-based NLP approach to search architectural knowledge entities in documents [FdRR12]. TREx is another approach that annotates architectural related documents by applying NLP to extract architectural knowledge entities (including architectural issue, drivers, and decision rationale) [LCAC12]. Although the operation stages of both approaches are similar to SADGE, the CoTs and the method of constructing CoTs are not presented in the referenced papers and not publicly accessible (e.g., on a project website). Therefore, it is not possible to replicate these two approaches, and as a result, a competitive quantitative comparison/evaluation or use on industry projects, is not feasible. The SADGE CoT and the methods of constructing CoT form two of our research contributions, in response to these limitations.

Contribution 3: Results of empirical evaluation of developing architectural decision guidance by employing a rule-based framework

Astudillo et al. [AVB12] has conducted an empirical study evaluating the TREx approach. The experiment presented in that study has been conducted on 21 students and only 2 experts, while this thesis presented results of an experiment with 21 experts from five different organizations besides 19 IT students. Furthermore, the authors of the mentioned paper have not presented the material of the experiment nor the development of reference document, and therefore replication is not possible.

5.2 Threats to Validity of Research Findings

This sections discusses the potential threats to the validity of the research results presented in this thesis. The section is grouped into internal validity and external validity.

5.2.1 Internal Validity

Exploratory Study. The researcher was the data collection instrument in the exploratory study (qualitative interviews). Therefore, one potential threat to internal validity of the explorartory study is the researcher's bias and his potentially various behaviors in different interview sessions. The researcher used an interview guideline including a list of important questions to be asked during all interview sessions, to undermine the effect of this threat.

Case Study. One shortcoming of the case study is that just one of the authors has annotated the reference documents, more than one opinion would be helpful to ensure the internal validity. The same author has annotated the reference documents in a second path, to decrease a possibly negative effect of this shortcoming.

Experiments. In both experiments, the potential threat to the internal validity is the

testing effect [CSG63]. To avoid the issue, the participants were divided into two groups and the two documents were swapped between the groups. As a result, group 1 in the second stage examined the sentences from the document that group 2 had in the first stage and vice versa.

Survey. The demographic questions in the beginning of the survey are designed to identify invalid answers so that these answers can be excluded from the analysis. Also a manual checking over the answers, with the focus on qualitative written answers is applied to find the possible spam answers. However, the chance that some irrelevant participants have given answers or that some participants have become careless in their response in the last scenarios of survey is not zero; which is a potential threat to internal validity.

5.2.2 External Validity

Exploratory Study. Generalization of results to all large-scale enterprise based on the enterprises from one industrial domain, is arguable. Although large-scale enterprises from other domains like telecommunication, finance, or health-care also have challenges on making architectural decisions for enterprise application development, our study shows that the software integration in the electricity industry is more immature than other areas due to lack of standardization and it can affect the architectural decision issues. So we are aware of the threat to external validity. Conducting the same interviews with enterprises from other domains would make the results more reliable.

Case Study. External validity is a goal that is more difficult to attain in a single case study [Tel97]. However, to increase the statistical generalizability of the results [Yin14], we chose three large and architecturally significant documents among six documents we received from the telecommunication company. The three documents belong to three different projects, and the authors of the first document are different from the authors of the other two documents. These selections ensured that the reference documents contained a sufficient amount of architectural issues, and have various language styles. Receiving more architecture-related documents from more companies would definitely increase the external validity of the case study. We requested documents from more companies (inside and outside Norway). Either they did not have available architecture-related documents in English, or it was not possible for them to share it with us due to confidentiality concerns.

Experiments. In both experiments, the potential threat to the external validity is the selection of the material (number, size and domain) for the experiment. We were aware of this issue, and therefore in the second experiment we tried to select text from two documents of different types (one is domain literature from Smart Grids, and the other is a document from a telecommunication company). The portions of the documents are selected in a way that all positive, negative, false positive, and false negative sentences are present in the text, as explained in Paper 4. We would need to ask the experts to spend

much more time on the experiment, to evaluate the framework by applying it on larger documents from more diverse domains, which was not feasible.

Survey. Small number of respondents is often a potential threat to external validity of surveys. However, the required number of raters in inter-rater agreement studies has been suggested differently, to ensure an adequate precision in the results [Gwe10, LS07]. Even when coefficient of variation for percent agreement is anticipated to be 5%, 40 raters are enough to participate in the study, according to Gwet [Gwe10].

5.2.3 Construct Validity

Experiments. One potential threat to construct validity of the experiments is that participants might interpret an ”architectural issue” differently from researchers. In that case, what the participants annotated as architectural issues in the experiment material might not be what the researchers meant by architectural issue. To undermine this threat, an introduction stage was included in both experiments to define an ”architectural issue” with some concrete examples.

Survey. Similarly, a potential threat to construct validity of the survey is possibly inconsistence interpretations of ”architectural issue” and ”quality attribute” between researchers and participants. The quality attributes were defined in the introduction of the survey based on a well-known standard quality model. Architectural issue was also defined and explained in the introduction of the survey. Besides, the survey was first conducted on some experts and their feedbacks were applied in designing the final version of the survey questionnaire to undermine the threat to construct validity.

5.2.4 Conclusion Validity

Experiments. The results of the experiment with students was not analyzed by any statistical methods and therefore the conclusion we draw from the first experiment has a potential threat to validity. To diminish this threat in the second experiment, we applied statistical methods to test both the normality of data distribution and the reliability of the comparison between results of the three extraction methods.

Survey. The survey could have a threat to conclusion validity if no reliability test was conducted on the data. ICC and Krippendorff’s alpha tests take the reliability into account besides measuring the agreement, to increase the conclusion validity.

5.3 Potential Impact on Practice

The discussion with practitioners at the end of the second experiment, shows that the potential value of semi-automatic architectural issue extraction generally is appreciated. Besides, during the design and evaluation of the SADGE framework, we received interest from companies in two European countries. Even though, projects employing agile practices might not see the need for systematic knowledge extraction/processing and decision making; they might assess the effort to overshadow the potential gains and follow the lean principle to "defer decisions to the last responsible moment" [PP03]. However, even extremely agile projects will not want the last responsible moment for decision making to morph into the least responsible moment. Hence, thought leaders and architects nowadays see architecture and agility as natural companions [ABK10, BNO10]. Rule-based issue extraction as in SADGE, can be leveraged in architectural spikes and early sprints to scope the fundamental design work that is required to mitigate technical risk. Thus, the project team can iterate often and sprint through the project once the baseline architecture is stable enough. This was pointed out by the experts participated in the second experiment.

A discussion that may be raised is the usefulness of reusing architectural decisions. Software development and architecture design methods such as Attribute-Driven Design and the Unified Process (UP), put requirements first, quality attributes in particular (rightfully so). Hence, it might not be obvious that decisions identified, made, and documented on previous projects should be considered at all. However, the study of reference architectures, patterns, and other reusable assets certainly has its place in the practitioner's toolbox; some methods used in industry even make this activity explicit. Finding a documented architectural decision in a project document does not mean (by any means) that this decision has to be followed blindly on the current project. However, the need for architectural issues, and the chosen and neglected design alternatives as well as the rationale for their inclusion to or exclusion from the architecture, are still highly informative, even if the context, requirements, and constraints of the current project differ from those of the first project. Hence, architectural decision reuse as supported by SADGE, can be seen as a natural extension of (and contribution to) such methods.

The findings of the survey show a poor agreement among experts in associating architectural issues with quality attributes. This informs the practitioners that the quality attributes they consider irrelevant or less relevant for an architectural issue, might be very relevant in the opinion of other practitioners. Hence, their favorite solution might have proven disadvantages connected to disregarded quality attributes. Exploring the decisions have been made for the same architectural issues by other practitioners in the same domain is one possible preventive solution, as it is facilitated by SADGE. Besides, group decision-making (GDM) and approaches and tools that support GDM [RM14, TGL+16] can be employed, especially for critical architectural issues. GDM brings more than one opinion into account and therefore the possible ignored yet relevant quality attributes by individ-

ual decision makers can be identified. Another solution is employing systematic mapping of architectural issues and quality attributes [GCSY08, KTS+09] or quantitative quality evaluation techniques [Koz12].

5.4 Strengths and Weaknesses of Solution

The rule-based framework presented in this thesis, like any other solution, has strengths and weaknesses. The strengths may encourage practitioners to use the framework and the researchers to extend the framework. The weaknesses, on the other hand, may make the potential users aware of the limitations of the framework and the researchers interested to improve the framework.

Strengths. Agile and lean software development are gaining more adoption in the industry. Software and enterprise architecture, in the meanwhile, are also finding their position as necessary properties of projects and organizations. While agility and architecture-aware development might sometime be perceived contradictory, SADGE has the potential to bring them into alignment. The framework treats recurring architectural issues as first class entities and rapidly recovers them from documentation of previous projects and related literature. Such a treatment gives the recurring architectural issues an active role in the decision-making process, and reduces the chance of occurring cognitive biases in decision makers.

Another strength of SADGE lies in its ad-hoc construction method for developing the Catalogue of Terms (CoT). This ability makes the framework adoptable for extracting architectural issues from documents in any company, within any context (telecommunication, banking, military, public sector, etc.). Besides, the default version of CoT contains some general terms and some software architectural terms. The general terms might be reusable for developing rule-based frameworks for extracting issues (decision topics) in other domains besides software architecture (e.g. management, economy, politics, etc.)

Weaknesses. The major strength of SADGE, extracting architectural issues from documents in a rapid way, comprises a potential weakness: eliminating the context. The Automatic Annotator in SADGE (which is built based on GATE) has a user interface where practitioners can see the whole text and the annotated sentences in the text. This feature enables the practitioners to see the context around each annotated sentence too. However, if the practitioners read the whole context surrounding a sentence, it is against the agility of the framework. On the other hand, if they focus only on annotated sentences they may miss some knowledge in the context, which is implicitly related to the annotated sentence. Therefore, users of SADGE should be aware of this trade-off.

Another weakness of SADGE is that it treats all architectural issues with a general attitude. Architectural issues can be divided into different categorizes based on decision lev-

els (conceptual, technology, or vendor asset), or can be associated with different decision drivers (security, performance, reliability, usability, etc.). For example, if a practitioner is a security architect concerned with only security-related decisions, applying SADGE on documents extracts all kinds of architectural issues besides security related ones, which (s)he might not be interested in.

CHAPTER 6

Conclusion

This thesis presents the results of several studies in support of improving architectural decision making processes: The first study has explored the general attitude of large-scale enterprises in architectural decision-making. The second study has investigated available tools and research prototypes that supports enhancing architectural decision guidance. The third and fourth studies have iteratively proposed and evaluated a rule-based framework for enhancing architectural decision guidance and presented the preliminary opinion of expert architects about the framework. The last study has investigated the inter-rater agreement among experts in associating architectural issues to quality attributes. This chapter revisits the research questions of the thesis and summarizes the key research findings that inform each research question.

RQ1: What is the attitude of large-scale enterprises in making and reusing architectural decisions and how do available tools and research prototypes support them?

In line with a few previous empirical studies, our exploratory study shows that most of the enterprises are not using well-known approaches such as ATAM. They are rather using their own structured procedures. The study also revealed that the relationships among the actors of a software ecosystem could significantly affect their architectural-decision making processes, for example by limiting their alternative solutions. This factor should also be considered as an influencing factor on architectural decision making process, in addition to the factors previous studies have extracted from the industry. The exploratory study also shows that there is a high potential among enterprises to reuse architectural decisions across various projects, or across different companies within a software ecosystem.

The results of the survey show that there is a poor agreement among experts on associating architectural issues with quality attributes. The study, therefore, suggests that practitioners pay more attention to systematic association of relevant quality attributes with an architectural issue, when they make decisions for solving the issue. Such systematic approaches already exist in the literature, yet are not widely adopted by the agile software development processes due to time related concerns.

The literature review we conducted, reports that the available tools and research prototypes have made significant contributions in the area of capturing architectural knowledge , but still require a number of extensions so that the captured knowledge can be enhanced to architectural decision guidance in practice.

RQ 2: How can a framework be established to develop architectural decision guidance from architecture-related documents in a rapid way?

Automatically extracting architectural issues from architectural documents (project documents and domain literature) reduces the effort of manually creating architectural guidance. Machine learning can be recognized as the first option, but it is not applicable in the architectural knowledge domain due to limited access to architecture-relaed documents written in natural language text. Alternatively, by employing natural language processing (NLP) approaches, we show that a rule-based framework can be employed for enhancing architectural guidance from architecture-related documents in a rapid way.

Applying the engineering method, we developed a framework called Semi-Automated Design Guidance Enhancer (SADGE). SADGE contains a rule-based information extractor that automatically extracts architectural issues from architecture-related documents. It uses a Catalogue of Terms (CoT) to realize which sentences of a document have certain terms. These terms are clues to discriminate a sentence as an architectural issue. We proposed an ad-hoc manual bootstrapping method for constructing catalogue-of-terms combined with a lexical database assistance approach. We developed a default version of CoT which should be adopted to the context of each document before launching SADGE on the document. Automatic extractor combined with manual CoT constructor, make the framework semi-automatic.

RQ 3: How efficient and effective will such a framework be in developing architectural decision guidance?

The results of the case study showed that SADGE extracts architectural issues with a significant recall, while keeping the effort reduction within an acceptable range. The results of the experiments with both students and expert architects show that the automatic extraction of architectural issues has higher recall and effort reduction compared with the semi-automatic approach. Therefore, considering the two evaluation metrics, the automatic approach is shown to be more efficient and effective than the semi-automatic approach for extracting architectural issues.

The general feedback from experiment participants show that the experts believe that

the framework can be very helpful for them to either reduce the amount of the text to read, or to determine the hot spots of their documents they need to pay extra attention to. Therefore, practitioners can use SADGE especially in the first stage of their architectural decision making process to rapidly identify architectural issues (decisions required) that are relevant to their project. This helps them accelerate the orientation into the problem-solution space, and consequently, to make architectural decisions in a more confident way and to be more prepared for mitigating the risks that occur during the development and evolution stages.

CHAPTER 7

Future Work

This chapter outlines different avenues of future research:

Supporting more natural languages. At the present, SADGE only works for documents written in English (no difference is made between American and British English). However, many projects around the world document their architectures in other languages (sometimes this is a conscious decision, but oftentimes it is a constraint imposed by certain stakeholders, e.g. corporate guidelines). Not only the Catalogue of Terms (CoT) will have to be adopted in this case, but also language grammars and character sets (e.g., in Asian languages) might have an impact on the framework implementation and configuration.

Developing domain-specific Catalogues of Terms. As our exploratory study show, the required software systems for large-scale enterprises are developed within a software ecosystem (SECO) rather than through a traditional isolated development fashion. Actors of a SECO, in spite of differences, may share some common concerns and requirements. Therefore, there would be recurring architectural issues in their software development or integration projects. For example, several electricity grid utilities in Norway should make similar architectural decisions in the direction of evolving their grids into Smart Grids. On the other hands, several software consultancy companies in Norway should make similar decisions for grid utilities as their customers. Therefore, one possible direction for future research is to give the current version of SADGE (as an open source package) to actors of a SECO and see how their ad-hoc CoT would evolve during a time slot (e.g. one year). Although, as we have faced it already, companies probably would not desire to share their architectural text with others (including researchers), the terms each company or organi-

zation adds to CoT for extracting architectural issues from their documents, can be asked to be accessible for researchers. In that case, common indicator terms for identifying architectural issues within a domain can be identified and new versions of CoT can be delivered for each domain (e.g. CoT for telecommunication, CoT for Smart Grids, CoT for finance, etc.).

Creating data sets for training and testing information extraction frameworks. As we showed in this thesis, one of the limitations that researchers face in developing information extraction frameworks in the architectural knowledge domain, is lack of data sets that include instances of architectural knowledge entities (architectural issues in our case). One direction for future study is creating and enriching a data set for training and evaluating frameworks that extract architectural knowledge entities from architectural documents written in natural language text.

Deeper investigation of architects' rationale for making architectural decisions. Poor inter-agreement between experts in associating architectural issues with quality attributes, made us interested to find out why different expert architects have different opinions in making architectural decisions. However, deeper investigation was out of the agenda of our research. So, a possible direction for future study is to conduct more exploratory studies (both qualitative and quantitative) to find answers.

Bibliography

[AB14] Steve Abelman and Randy Bass. Observed heuristics and biases in air traffic management decision making using convective weather uncertainty. 2014.

[ABK10] Pekka Abrahamsson, Muhammad Ali Babar, and Philippe Kruchten. Agility and architecture: Can they coexist? *Software, IEEE*, 27(2):16–22, 2010.

[ACC12] Mohsen Anvaari, Daniela S Cruzes, and Reidar Conradi. Challenges on software defect analysis in smart grid applications. In *Proceedings of the First International Workshop on Software Engineering Challenges for the Smart Grid*, pages 19–22. IEEE Press, 2012.

[Álv13] Carlos García Álvarez. Overcoming the limitations of agile software development and software architecture. 2013.

[AVB12] Hernan Astudillo, Gonzalo Valdes, and Carlos Becerra. Empirical measurement of automated recovery of design decisions and structure. In *Andean Region International Conference (ANDESCON), 2012 VI*, pages 105–108. IEEE, 2012.

[Bar66] John J Bartko. The intraclass correlation coefficient as a measure of reliability. *Psychological reports*, 19(1):3–11, 1966.

[Bas93] Victor R Basili. The experimental paradigm in software engineering. In *Experimental Software Engineering Issues: Critical Assessment and Future Directions*, pages 1–12. Springer, 1993.

[BB11] Alan Bryman and Emma Bell. *Business Research Methods 3e*. Oxford university press, 2011.

[BBM13] Muhammad Ali Babar, Alan W Brown, and Ivan Mistrík. *Agile Software Architecture: Aligning Agile Processes and Software Architectures*. Newnes, 2013.

[BCK03] Len Bass, Paul Clements, and Rick Kazman. *Software Architecture in Practice*. Addison-Wesley Longman Publishing Co., Inc., 2003.

[BDLvV09] Muhammad Ali Babar, Torgeir Dingsøyr, Patricia Lago, and Hans van Vliet. *Software architecture knowledge management*. Springer, 2009.

[Bie10] Matthias Biehl. Literature study on design rationale and design decision documentation for architecture descriptions. Technical report, Technical Report ISRN/KTH/MMK, 2010.

[BJ08] Pamela Baxter and Susan Jack. Qualitative case study methodology: Study design and implementation for novice researchers. *The qualitative report*, 13(4):544–559, 2008.

[BNO10] Nanette Brown, Robert Nord, and Ipek Ozkaya. Enabling agility through architecture. Technical report, DTIC Document, 2010.

[Bri95] Nicky Britten. Qualitative research: qualitative interviews in medical research. *Bmj*, 311(6999):251–253, 1995.

[BSJ09] Sjaak Brinkkemper, Ivo van Soest, and Slinger Jansen. Modeling of product software businesses: Investigation into industry product and channel typologies. In Wita Wojtkowski, Gregory Wojtkowski, Michael Lang, Kieran Conboy, and Chris Barry, editors, *Information Systems Development*, pages 307–325. Springer US, 2009.

[Bur91] Philip Burnard. A method of analysing interview transcripts in qualitative research. *Nurse education today*, 11(6):461–466, 1991.

[CD11] Daniela S Cruzes and Tore Dybå. Recommended steps for thematic synthesis in software engineering. In *Empirical Software Engineering and Measurement (ESEM), 2011 International Symposium on*, pages 275–284. IEEE, 2011.

[CGC10] Agustin Casamayor, Daniela Godoy, and Marcelo Campo. Identification of non-functional requirements in textual specifications: A semi-supervised learning approach. *Information and Software Technology*, 52(4):436–445, 2010.

[CMRW77] General Electric Company, Jim A McCall, Paul K Richards, and Gene F Walters. *Factors in Software Quality: Final Report*. Information Systems Programs, General Electric Company, 1977.

[Con14] Thomas E Conine. Estimating the probability of meeting financial commitments: A behavioral finance perspective based on business simulations. *Global Business and Organizational Excellence*, 33(2):6–13, 2014.

[CSG63] Donald Thomas Campbell, Julian C Stanley, and Nathaniel Lees Gage. Experimental and quasi-experimental designs for research. Technical report, Houghton Mifflin Boston, 1963.

[Cun92] Ward Cunningham. The wycash portfolio management system, addendum to the proceedings on object-oriented programming systems, languages, and applications (addendum), 1992.

[Cun02] Hamish Cunningham. Gate, a general architecture for text engineering. *Computers and the Humanities*, 36(2):223–254, 2002.

[CW14] Erik Cambria and Bebo White. Jumping nlp curves: A review of natural language processing research. *IEEE Computational Intelligence Magazine*, 9(2):48–57, 2014.

[Cyb96] Jacob L Cybulski. Introduction to software reuse. 1996.

[EKGER] GH El-Khawaga, Galal Hassan Galal-Edeen, and AM Riad. Architecting in the context of agile software development: fragility versus flexibility.

[FCKK11] Davide Falessi, Giovanni Cantone, Rick Kazman, and Philippe Kruchten. Decision-making techniques for software architecture design: A comparative survey. *ACM Computing Surveys (CSUR)*, 43(4):33, 2011.

[FdRR12] Adriana Maria Figueiredo, Julio C dos Reis, and Marcos A Rodrigues. Improving access to software architecture knowledge an ontology-based search approach. *International Journal Multimedia and Image Processing (IJMIP)*, 2(1/2), 2012.

[FK05] William B Frakes and Kyo Kang. Software reuse research: Status and future. *IEEE transactions on Software Engineering*, (7):529–536, 2005.

[GAO95] D Garlan, R Allen, and J Ockerbloom. Architectural mismatch: why reuse is so hard. *Software, IEEE*, 12(6):17–26, 1995.

[GCSY08] Gokhan Gokyer, Semih Cetin, Cevat Sener, and Meltem T Yondem. Nonfunctional requirements to architectural concerns: Ml and nlp at crossroads. In *Software Engineering Advances, 2008. ICSEA'08. The Third International Conference on*, pages 400–406. IEEE, 2008.

[GKN+07] Richard Gabriel, R. Kazman, Linda Northrop, D. Schmidt, and K. Sullivan. Workshop on software technologies for ultra-large scale systems. In *Software Engineering - Companion, 2007. ICSE 2007 Companion. 29th International Conference on*, pages 140–141, May 2007.

[GLVV10] Qing Gu, Patricia Lago, and Hans Van Vliet. A template for soa design decision making in an educational setting. In *Software Engineering and Advanced Applications (SEAA), 2010 36th EUROMICRO Conference on*, pages 175–182. IEEE, 2010.

[Gro05] Object Management Group. *Reusable Asset Specification (RAS) Version 2.2*. 2005.

[Gru13] John Grundy. Architecture vs agile: competition or cooperation?, 2013.

[GVCC98] Julio Gonzalo, Felisa Verdejo, Irina Chugur, and Juan Cigarran. Indexing with wordnet synsets can improve text retrieval. *arXiv preprint cmp-lg/9808002*, 1998.

[Gwe10] Kilem L Gwet. The definitive guide to measuring extent of agreement among multiple raters, 2010.

[HAZ07] Neil B Harrison, Paris Avgeriou, and Uwe Zdun. Using patterns to capture architectural decisions. *Software, IEEE*, 24(4):38–45, 2007.

[HEK15] Sebastian Hanschke, Jan Ernsting, and Herbert Kuchen. Integrating agile software development and enterprise architecture management. In *System Sciences (HICSS), 2015 48th Hawaii International Conference on*, pages 4099–4108. IEEE, 2015.

[HFLVV11] Johan F Hoorn, Rik Farenhorst, Patricia Lago, and Hans Van Vliet. The lonesome architect. *Journal of Systems and Software*, 84(9):1424–1435, 2011.

[HKW04] Wiebe Hordijk, Dennis Krukkert, and Roel Wieringa. The impact of architectural decisions on quality attributes of enterprise information systems: a survey of the design space. 2004.

[IA10] Ana Ivanović and Pierre America. Information needed for architecture decision making. In *Proceedings of the 2010 ICSE Workshop on Product Line Approaches in Software Engineering*, pages 54–57. ACM, 2010.

[ISO11a] ISO ISO. IEC25010: 2011 systems and software engineering–systems and software quality requirements and evaluation (square)–system and software quality models. *International Organization for Standardization*, 2011.

[ISO11b] ISO/IEC/IEEE. ISO/IEC/IEEE systems and software engineering – architecture description. *ISO/IEC/IEEE 42010:2011(E) (Revision of ISO/IEC 42010:2007 and IEEE Std 1471-2000)*, pages 1–46, Dec 2011.

[JB05] A. Jansen and J. Bosch. Software architecture as a set of architectural design decisions. In *Software Architecture, 2005. WICSA 2005. 5th Working IEEE/IFIP Conference on*, pages 109–120, 2005.

[JBA08] Anton Jansen, Jan Bosch, and Paris Avgeriou. Documenting after the fact: Recovering architectural design decisions. *Journal of Systems and Software*, 81(4):536–557, 2008.

[JBF09] Slinger Jansen, Sjaak Brinkkemper, and Anthony Finkelstein. Business network management as a survival strategy: A tale of two software ecosystems. 2009.

[JSS14] R Jayasudha, S Subramanian, and L Sivakumar. A novel software reuse method-an ontological approach. *International Journal of Scientific and Engineering Research*, 2014.

[Kah11] Daniel Kahneman. *Thinking, fast and slow*. Macmillan, 2011.

[Kas05] Mark Kasunic. Designing an effective survey. Technical report, DTIC Document, 2005.

[Kav96] Steinar Kavle. Interviews. an introduction to qualitative research interviewing. *Interviews: an introduction to qualitative research interviewing*, 1996.

[KH01] JWKJW Kotrlik and CCHCC Higgins. Organizational research: Determining appropriate sample size in survey research appropriate sample size in survey research. *Information technology, learning, and performance journal*, 19(1):43, 2001.

[KKB+98] Rick Kazman, Mark Klein, Mario Barbacci, Tom Longstaff, Howard Lipson, and Jeromy Carriere. The architecture tradeoff analysis method. In *Engineering of Complex Computer Systems, 1998. ICECCS'98. Proceedings. Fourth IEEE International Conference on*, pages 68–78. IEEE, 1998.

[Klü15] Peter Klügl. *Context-specific Consistencies in Information Extraction: Rule-based and Probabilistic Approaches*. BoD–Books on Demand, 2015.

[KM05] Katharina Kaiser and Silvia Miksch. Information extraction. 2005.

[KNO12] Philippe Kruchten, Robert L Nord, and Ipek Ozkaya. Technical debt: from metaphor to theory and practice. *IEEE Software*, (6):18–21, 2012.

[Koz12] Anne Koziolek. Research preview: Prioritizing quality requirements based on software architecture evaluation feedback. In *Requirements Engineering: Foundation for Software Quality*, pages 52–58. Springer, 2012.

[Kri07] Klaus Krippendorff. Computing krippendorff's alpha reliability. *Departmental Papers (ASC)*, page 43, 2007.

[Kru92] Charles W Krueger. Software reuse. *ACM Computing Surveys (CSUR)*, 24(2):131–183, 1992.

[Kru13] Philippe Kruchten. Games software architects play. 2013.

[KTS+09] Haruhiko Kaiya, Masaaki Tanigawa, Shunichi Suzuki, Tomonori Sato, and Kenji Kaijiri. Spectrum analysis for quality requirements by using a term-characteristics map. In *Advanced Information Systems Engineering*, pages 546–560. Springer, 2009.

[LA09] Peng Liang and Paris Avgeriou. Tools and technologies for architecture knowledge management. In *Software Architecture Knowledge Management*, pages 91–111. Springer, 2009.

[LAL15] Zengyang Li, Paris Avgeriou, and Peng Liang. A systematic mapping study on technical debt and its management. *Journal of Systems and Software*, 101:193–220, 2015.

[Lan07] Jo Ann Lane. Understanding differences between system of systems engineering and traditional systems engineering. Technical report, 2007.

[LBJH12] Garm Lucassen, Sjaak Brinkkemper, Slinger Jansen, and Eko Handoyo. Comparison of visual business modeling techniques for software companies. In *Software Business*, pages 79–93. Springer, 2012.

[LCAC12] Claudia López, Víctor Codocedo, Hernán Astudillo, and Luiz Marcio Cysneiros. Bridging the gap between software architecture rationale formalisms and actual architecture documents: An ontology-driven approach. *Science of Computer Programming*, 77(1):66–80, 2012.

[LLA14] Zengyang Li, Peng Liang, and Paris Avgeriou. Architectural debt management in value-oriented architecting. *Economics-Driven Software Architecture, Elsevier*, pages 183–204, 2014.

[LLA15] Zengyang Li, Peng Liang, and Paris Avgeriou. Architectural technical debt identification based on architecture decisions and change scenarios. In *12th Working IEEE/IFIP Conference on Software Architecture*, 2015.

[LS07] James M LeBreton and Jenell L Senter. Answers to 20 questions about interrater reliability and interrater agreement. *Organizational Research Methods*, 2007.

[Mai98] Mark W. Maier. Architecting principles for systems-of-systems. *Systems Engineering*, 1(4):267–284, 1998.

[MASS08] Mehdi Mirakhorli, Amir Azim Sharifloo, and Fereidoon Shams. Architectural challenges of ultra large scale systems. In *Proceedings of the 2Nd International Workshop on Ultra-large-scale Software-intensive Systems*, ULSSIS '08, pages 45–48, New York, NY, USA, 2008. ACM.

[MBM13] Hendrik Meth, Manuel Brhel, and Alexander Maedche. The state of the art in automated requirements elicitation. *Information and Software Technology*, 55(10):1695–1709, 2013.

[Mil95] George A Miller. Wordnet: a lexical database for english. *Communications of the ACM*, 38(11):39–41, 1995.

[MMR14] José P Miguel, David Mauricio, and Glen Rodríguez. A review of software quality models for the evaluation of software products. *arXiv preprint arXiv:1412.2977*, 2014.

[Net09] Microsoft Developer Network. Microsoft application architecture guide, 2nd edition. `http://msdn.microsoft.com/en-us/library/ff650706.aspx`, 2009.

[NFG+06] L. Northrop, P. Feiler, R. P. Gabriel, J. Goodenough, R. Linger, T. Longstaff, R. Kazman, M. Klein, D. Schmidtd, K. Sullivan, and K. Wallnau. Ultra-Large-Scale Systems: The Software Challenge of the Future. Technical report, Sofwtare Engineering Institute, Carnegie-Mellon, 2006.

[PD12] Christian R Prause and Zoya Durdik. Architectural design and documentation: Waste in agile development? In *Software and System Process (ICSSP), 2012 International Conference on*, pages 130–134. IEEE, 2012.

[PP03] Mary Poppendieck and Tom Poppendieck. *Lean software development: an agile toolkit*. Addison-Wesley Professional, 2003.

[PPV00] Dewayne E Perry, Adam A Porter, and Lawrence G Votta. Empirical studies of software engineering: a roadmap. In *Proceedings of the conference on The future of Software engineering*, pages 345–355. ACM, 2000.

[RA10] B.R. Rad and F.S. Aliee. Computational grid as an appropriate infrastructure for ultra large scale software intensive systems. In *Complex, Intelligent and Software Intensive Systems (CISIS), 2010 International Conference on*, pages 469–474, Feb 2010.

[Rad22] George Stanley Radford. *The control of quality in manufacturing*. Ronald Press Company, 1922.

[RM14] Smrithi Rekha and Henry Muccini. Suitability of software architecture decision making methods for group decisions. In *Software Architecture*, pages 17–32. Springer, 2014.

[Rob11] Colin Robson. *Real world research: a resource for users of social research methods in applied settings*. Wiley Chichester, 2011.

[RR98] Jason E Robbins and David F Redmiles. Software architecture critics in the argo design environment. *Knowledge-Based Systems*, 11(1):47–60, 1998.

[RW03] Ellen Riloff and Janyce Wiebe. Learning extraction patterns for subjective expressions. In *Proceedings of the 2003 conference on Empirical Methods*

in Natural Language Processing, pages 105–112. Association for Computational Linguistics, 2003.

[RWP05] Ellen Riloff, Janyce Wiebe, and William Phillips. Exploiting subjectivity classification to improve information extraction. In *Proceedings of the National Conference On Artificial Intelligence*, volume 20, page 1106. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2005.

[Sar08] Sunita Sarawagi. Information extraction. *Foundations and trends in databases*, 1(3):261–377, 2008.

[SC13] Richard Mark Soley and Bill Curtis. The consortium for IT software quality (CISQ). In *Software Quality. Increasing Value in Software and Systems Development*, pages 3–9. Springer, 2013.

[Sha02] Mary Shaw. What makes good research in software engineering? *International Journal on Software Tools for Technology Transfer*, 4(1):1–7, 2002.

[Sha08] Mary Shaw. Empirical challenges in ultra large scale systems. In *Proceedings of the Second ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*, ESEM '08, pages 110–110, New York, NY, USA, 2008. ACM.

[SLK09] Mojtaba Shahin, Peng Liang, and Mohammad Reza Khayyambashi. Architectural design decision: Existing models and tools. In *Software Architecture, 2009 & European Conference on Software Architecture. WICSA/ECSA 2009. Joint Working IEEE/IFIP Conference on*, pages 293–296. IEEE, 2009.

[SM95] Webb Stacy and Jean MacMillan. Cognitive bias in software engineering. *Communications of the ACM*, 38(6):57–63, 1995.

[Sva04] Mikael Svahnberg. An industrial study on building consensus around software architectures and quality attributes. *Information and Software Technology*, 46(12):805–818, 2004.

[SW11] Claude Sammut and Geoffrey I Webb. *Encyclopedia of machine learning*. Springer Science & Business Media, 2011.

[TA05] Jeff Tyree and Art Akerman. Architecture decisions: Demystifying architecture. *IEEE software*, (2):19–27, 2005.

[TAJ+10] Antony Tang, Paris Avgeriou, Anton Jansen, Rafael Capilla, and Muhammad Ali Babar. A comparative study of architecture knowledge management tools. *Journal of Systems and Software*, 83(3):352–370, 2010.

[TBGH05] Antony Tang, Muhammad Ali Babar, Ian Gorton, and Jun Han. A survey of the use and documentation of architecture design rationale. In *Software*

Architecture, 2005. WICSA 2005. 5th Working IEEE/IFIP Conference on, pages 89–98. IEEE, 2005.

[Tel97] Winston M Tellis. Application of a case study methodology. *The qualitative report*, 3(3):1–19, 1997.

[TGL+16] Dan Tofan, Matthias Galster, Ioanna Lytra, Paris Avgeriou, Uwe Zdun, Mark-Anthony Fouche, Remco de Boer, and Fritz Solms. Empirical evaluation of a process to increase consensus in group architectural decision making. *Information and Software Technology*, 72:31–47, 2016.

[TK73] Amos Tversky and Daniel Kahneman. Availability: A heuristic for judging frequency and probability. *Cognitive psychology*, 5(2):207–232, 1973.

[TMD09] Richard N Taylor, Nenad Medvidovic, and Eric M Dashofy. *Software architecture: foundations, theory, and practice*. Wiley Publishing, 2009.

[vHA11] Uwe van Heesch and Paris Avgeriou. Mature architecting-a survey about the reasoning process of professional architects. In *Software Architecture (WICSA), 2011 9th Working IEEE/IFIP Conference on*, pages 260–269. IEEE, 2011.

[WA14] Claes Wohlin and Aybüke Aurum. Towards a decision-making structure for selecting a research design in empirical software engineering. *Empirical Software Engineering*, pages 1–29, 2014.

[Yin14] Robert K Yin. *Case study research: Design and methods*. Sage publications, 2014.

[Zim09] Olaf Zimmermann. *An architectural decision modeling framework for service oriented architecture design*. Doctoral thesis, University of Stuttgart, Faculty of Computer Science, Electrical Engineering, and Information Technology, Germany, March 2009.

[Zim11] Olaf Zimmermann. Architectural decisions as reusable design assets. *Software, IEEE*, 28(1):64–69, Jan 2011.

[ZKL+09] Olaf Zimmermann, Jana Koehler, Frank Leymann, Ronny Polley, and Nelly Schuster. Managing architectural decision models with dependency relations, integrity constraints, and production rules. *Journal of Systems and Software*, 82(8):1249–1267, 2009.

[ZM12] Olaf Zimmermann and Christoph Miksovic. Decisions required vs. decisions made: Connecting enterprise architects and solution architects via guidance models. *Aligning Enterprise, System, and Software Architectures*, page 176, 2012.

[ZMK12] Olaf Zimmermann, Christoph Miksovic, and Jochen M Küster. Reference architecture, metamodel, and modeling principles for architectural knowledge management in information technology services. *Journal of Systems and Software*, 85(9):2014–2033, 2012.

APPENDIX A

Selected Papers

P1- Architectural Decision-Making in Enterprises

Published: In Proc. The 7th European Conference on Software Architecture, ECSA 2013.

Architectural Decision-Making in Enterprises: Preliminary Findings from an Exploratory Study in Norwegian Electricity Industry

Mohsen Anvaari, Reidar Conradi, Letizia Jaccheri

Norwegian University of Science and Technology, Trondheim, Norway
{mohsena,conradi,letizia}@idi.ntnu.no

Abstract. Motivation: The current literature in the architectural knowledge domain has made a significant contribution related to documenting software architectural decisions. However, not many studies have been conducted to assess the architectural decision-making and decision reuse processes through empirical investigations. Besides, the effect of the relationships among the actors in a software ecosystem on the architectural decisions-making process of each actor is not well studied. **Goal:** The objective of this paper is to identify the main processes and issues on the architectural decision-making in large-scale enterprises by considering the relationships among the enterprises and other actors of the ecosystem. **Method:** We conducted semi-structured interviews with six Norwegian companies in the software ecosystem of electricity industry. **Results:** Regarding the architectural decision-making process, the findings are in line with previous empirical studies, showing that most of the companies are not using well-known academic approaches such as ATAM, they are rather using their own procedures. The study also shows that the relationships among the actors of a software ecosystem could significantly affect the architectural-decision making process in each of the actors, for example, by limiting their alternative solutions. Finally, the results confirm that it is advantageous for the enterprises to reuse the architectural decisions across their various projects or for cooperative companies to reuse the decisions across their similar projects. **Conclusion:** Improving the reusable architectural decision frameworks by considering the relationships among the actors in a software ecosystem would be beneficial for the industry.

Keywords: Architectural decision making, enterprise applications, empirical study, software ecosystem, electricity industry

1 Introduction

In the current industrial environments, enterprises [1] employ various software applications to automate their daily business processes and activities. They buy the applications from different vendors and use them either separately or as an integrated system based on a high level structure (architecture). Therefore the concepts such as *enterprise application* [2], *enterprise application development (EAD)*, *enterprise application integration (EAI)*, *enterprise architecture* and similar terms have been developed and used for many years in the both academia and industry.

[1] "An enterprise is one or more organizations sharing a definite mission, goals and objectives to offer an output such as a product or a service" (Chen et al., 2008).

[2] An enterprise application is a distributed, software-intensive system that automates business processes and activities in an enterprise (Zimmermann, 2009).

By evolving and expanding the usage of such systems, the amount of transactional data between different applications have been dramatically increased and as a result many enterprises automate the data transfer between their applications. Therefore a movement from software as an island to software as a systems-of-systems has been emerged for many years (Maier, 1998). A classical challenge in this landscape is integration and interoperability issue (Chen et al., 2008)(Fisher, 2006) because the applications are developed based on different platforms (programming languages, operating systems, network protocols, etc.). Many approaches, trends and standardizations have been introduced to decrease the interoperability challenges (Chen et al., 2008) but still interoperability is one of the main concerns in EAD.

One of the most successful approaches for solving the interoperability issue is service-oriented development and many enterprises are using service-oriented architecture (SOA) as their architectural style. Although SOA is shown to be highly successful to alleviate the interoperability problem, implementing a SOA in an enterprise is not easy and has to meet domain-specific non-functional requirements with explicit software quality criteria (Zimmermann et al., 2007). There are many ways of implementing a SOA and no single SOA fits all purposes and constraints of an organization. Therefore many architectural design issues and tradeoffs arise (ibid), and architects have hard time to make "right" architectural decisions. Such decisions would include strategic concerns about technology and product selection, finding the right service interface granularity, and numerous decisions that deal with non-functional aspects (ibid). Zimmermann et al. have captured 130 such SOA decisions till 2007. This shows how challenging is to choose between different decisions (ibid).

How do enterprises deal with complex integration and how do they make efficient architectural decisions? What are their main challenges (technical and organizational) to make the right decisions? Do the enterprises reuse their architectural decisions in their different but similar projects? What about software consultant companies, do they reuse architectural decisions in different enterprises in the same domain? How the relationships between different companies and organizations in an industrial domain would affect their architectural decisions?

Even though the *architectural decision* concept (and the broader concept, *architectural knowledge*[3]) has gained increasing attention in the software architecture community in the last decade, still there are some deficiencies in answering to the mentioned questions:

- Based on our literature review, existing works in the architectural knowledge are more theoretical frameworks and tools developed in the academia and very few empirical researches exist in the area. Even though the theoretical works have been often evaluated by industrial case studies, the assumptions and claims about architectural decision-making in enterprises are seldom obtained through

[3] Architectural knowledge = architectural decisions + architectural design (Kruchten et al., 2006).

empirical studies. The motivations for developing such frameworks and tools are mostly gained from either previous literature or authors' personal experiences in the industry. There is a lack on getting insights from the industry in a more systematic way.

- Even those few empirical studies in the area (we will try to cover them in the related work section) are mostly focused on the decision documentation and representing. The decision-making process in the industry has not been often studied empirically.
- Despite all these, still there are some empirical studies and surveys to understand the decision-making and reasoning process of architects in the industry. But first of all they are not discussing the reusable architectural decisions in EAD (Zimmermann, 2009). Furthermore they don't consider the effect of companies relationships on the decision-making process.

Considering the above issues, the goal of this paper is to get insights for answering to the mentioned questions by observing the current situation of software development and integration in the Norwegian electricity industry. The remainder of the papers is organized as follows: In the section 2 the related work will be discussed. Section 3 presents the design of the research including the research goal and questions and the research method. Section 4 presents the results of the study and section 5 analysis the resutls. Finally section 6 remarks the conclusions and also discusses the future work.

2 Related Work

To find out the related areas to this research, we consider three dimensions: topic of interest, research method and research context. The topic of our research is generally related to the architectural knowledge and more specifically the architectural decision area. If we divide the works in this dimension to the *making the decisions* and *documenting the decisions*, our work is focused more in the making the decisions. Concerning the research method, if we consider *theoretical-based researches* and *empirical investigations*, this research is related to the empirical investigations. Regarding the context, we split the current work into the research that studies the architectural decisions in a company regardless of its position in the software ecosystem (SECO), and research that considers the company position in SECO. Our work focus is on the latter.

2.1 Making Architectural Decisions

Making architectural decision is the process of selecting one alternative among different alternatives for solving a design issue in a software system (Falessi et al., 2011). As we mentioned earlier, this concept is a part of architectural knowledge and

has become increasingly important in the software architecture community since the beginning of 2000s. Many researchers have worked in this area and have discussed the importance of the decisions and the rationale behind the decisions. Many tools and frameworks have been developed by the researchers to support the practitioners in the activities around the architectural decisions. Babar et al. in their book that published in 2009 have reviewed and gathered many of the works that have been done in this area in the last decade (Babar et al., 2009). Tang et al. have also covered some of the existing architectural knowledge management tools (Tang et al., 2010).

Although the existing work in architectural knowledge area focuses more on documenting and representing the decisions, still there is some work that supports the decision-making process. For example Falessi et al. have reviewed and compared the available techniques and tools for making architectural decisions in their comparative survey (Faless et al., 2011). However, most of the existing frameworks and tools have a general approach and are not specified for the enterprise application development and integration (Zimmermann, 2009) that is the interest of this research. Furthermore, seldom they consider reusing the architectural decisions in the similar projects or domains while many issues recur in the enterprise projects and reusing the architectural decisions from previous projects would be helpful (Zimmermann, 2009). Although Falessi et al. have mentioned that reuse can help to simplify the architecting (Falessi et al., 2011) they have not considered it in their analysis. Zimmermann's work (Zimmermann, 2009) is actually a reusable architectural decision model in enterprise application development and integration and therefore is a source of inspiration for our work. Even though, he has not considered the effect of companies relationships on their architectural decisions.

Finally, as discussed earlier, the motivations and insights for developing frameworks, techniques and tools that support the decision making process in the industry have been gained mostly from researchers' personal experiences and not through a systematic empirical observation. Nevertheless, recently some few empirical studies have been conducted in this area that we will discuss them in the next part.

2.2 Empirical Studies

In software engineering research in general, without knowing the fundamental mechanisms that derive the costs and benefits of software tools and methods for a certain application, "we can't say whether we are basing our actions on faulty assumptions, evaluating new methods properly, and inadvertently focusing on low-payoff improvements. In fact, unless we understand the specific factors that cause tools and methods to be more or less cost-effective, the development and use of a particular technology will essentially be a random act. Empirical studies are a key way to get this

information and move towards well-founded decisions" (Perry et al., 2000). It is the same situation in the architectural decisions area and conducting empirical studies and observations would be the base for developing effective methods, frameworks and tools. Till 2005, there were little empirical evidence about architectural decisions and how practitioners treat them in the practice (Tang et al., 2005). Tang et al. conducted a survey on the use and documentation of architecture design rationale in 2005. Even though, their main focus was to understand how practitioners think about decision rationale, how they use and document them, and what factors prevent them from documenting decision rationale (ibid). Making the decisions were not the focus of their investigations. Hoorn et al. were interested in the same direction and did a broad survey to better understand what architects really do and what kind of support they need for sharing architectural knowledge (Hoorn et al., 2011).
Ivanovic and America has conducted a study to gain knowledge on information needed for architecture decisions made by architects and managers (Ivanovic and America, 2010). The reuse aspect of the decisions is not in their work. Also they have conducted their study only in one industrial organization and therefore considering the ecosystem relationships that we are interested in is not in their research.
Finally, van Heesch and Avgeriou in their study have investigated how experienced architects reason in the context of industrial projects, how they prioritize the problem space, how they propose solutions for the problem and how they choose among solutions (van Heesch and Avgeriou, 2011). Their work is relevant to our research topic but still lacks the reuse aspect and also has not considered software ecosystem relationships. In the next part we will explain the software ecosystem concept and what we mean by a software ecosystem relationship and how we want to illustrate the software ecosystem of our research context.

2.3 Software Ecosystem

A software ecosystem (SECO) is "a set of actors functioning as a unit and interacting with a shared market for software and services, together with the relationships among them" (Jansen et al., 2009). The actor type in a SECO could be supplier, independent software vendor, software consulting company or intermediary, and customer (Brinkkemper et al., 2009). Interaction or relationship type could be product flow, service flow, financial flow and content flow (ibid). There are several ways to model and illustrate a software supply network (SSN) within a SECO (Lucassen et al., 2012). In the section 3.2, to illustrate the SSN of our research context that is Norwegian electricity industry, we have created a figure that is based on the model by (Brinkkemper et al., 2009). Since we are interested to see how the SECO relationships would affect the architectural decisions, some customizations have been made to the model to fit to our context and intentions. To our best knowledge there is no empirical

study that has considered the effect of SECO relationships on the architectural decision processes.

3 Research Design

Our investigation is an exploratory study (Robson, 2011) which aims to identify the situation in a real world context. Qualitative data is collected by interviews and analyzed using thematic synthesis. In the following sections we explain the research questions, the context of the study, data collection and analysis methods and threats to results validity.

3.1 Goal

The goal of this research is to identify the main processes and issues on making and reusing architectural decisions in large-scale enterprises by considering the relationships among the enterprises and other actors of the ecosystem. To reach to the goal we are interested to find out:

RQ1. How do industrial companies make architectural decisions for enterprise application development with respect to decision-making methodology?
RQ2. How do the companies reuse the architectural decisions in various projects?
RQ3. How do the software ecosystem relationships affect the decision making process?

By RQ1 we aim to explore the general attitude of companies in making their significant architectural decisions. Such decisions would include the high-level blueprint of their software and information systems to the detailed technical decisions such as choice of integration platform. Although the previous research studies had explored this aspect (van Heesch and Avgeriou, 2011), we investigate the answer to RQ1 by the means of qualitative observations to find out the possible uncover aspects of decision-making process in companies.
The aim of RQ2 is to discover whether companies reuse their architectural decisions in different projects and if the answer is no to investigate if it is possible to do so or not.
RQ3 considers the relationships among various actors in the software ecosystem and its possible effect on the process of making and reusing architectural decisions in each actor.

3.2 Context

Since this research is contextualized in a larger research project on *software engineering support for Smart Grid,* our main case of the study is the Norwegian electricity industry.

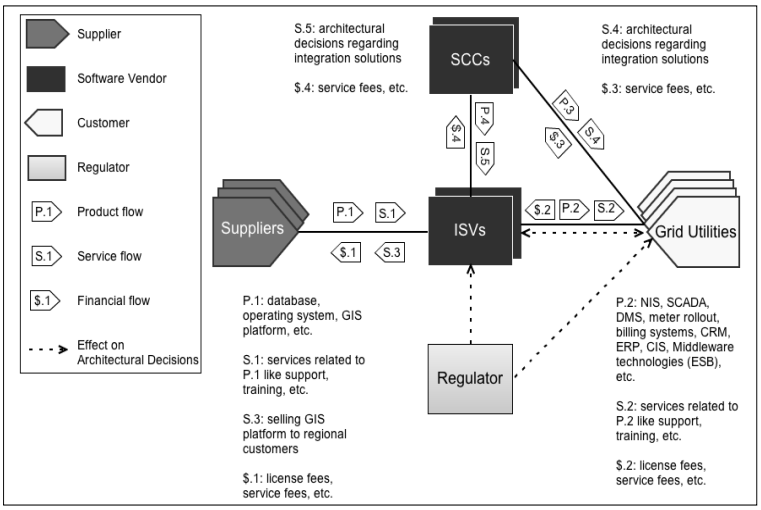


Fig. 1 Current software supply network in the Norwegian electricity industry

The software market in this industry, the same as other domains, has become a software ecosystem including different actors and various relationships between them. The actors make a software supply network to develop and integrate the required software products. Fig. 1. shows the current state of software supply network in the Norwegian electricity industry. It is based on the result of a previous interview that we conducted with an expert in the Norwegian electricity industry and later confirmed by some other experts. The chain of supply network could differ among various customers based on their size and organizational policies, but a typical path can be described as follows: A grid utility (customer) needs different software products to run its daily business activities. The utility negotiates with different national and international independent software vendors (ISV). The ISVs themselves should buy some of their fundamental software components and packages (OS, DB, etc.) from their suppliers and develop their products based on the provided components. Grid utilities later on integrate various software products themselves or ask software consultant companies (SCC) to do it for them. Sometimes ISVs also ask SCCs to help them regarding the software development to produce more interoperable solutions. There is also a national regulator that although doesn't deliver software product or service to the customers but affect the software development in ISVs and software integration in grid utilities by the rules and regulations. So we have added "regulator" as a new actor to the model from (Brinkkemper et al., 2009). Also to show how the SECO relationships would affect the architectural decisions (see sections 4.4 and 5.2), we have added a new object to the model (the dashed arrow).

3.3 Data Collection and Analysis Methods

To answer to the research questions (see section 3.1) the semi-structured interview has been chosen as the main data collection method of this research. As it was discussed in the context part, our target companies lie in four categories based on their role in the software ecosystem of Norwegian electricity industry: grid utilities, software vendors, software consultant companies and regulator. Our initial plan was to select different samples from each of these categories. Currently there are almost 150 grid utilities in Norway, but the software vendors and also the software consultant companies that are delivering products and service to them are very few and it makes the sample selection challenging. For this stage of our research we conducted interview with 5 grid utilities (all of them have more than 75.000 grid customers) and one software vendor that have 80 percent of market share in Norway. We couldn't convince either the regulator or any software consultant companies that have experiences with software for electricity industry to participate in our study within out time frame.
To prepare the questionnaire, initially we selected 10 questions regarding the decision making process that applies to all categories of companies regardless of their role in the SECO. Some of the questions were inspired by the questionnaire van Heesch and Avgeriou have used in their survey (van Heesch and Avgeriou, 2011). The preliminary set of questions were written as follows:

1. A brief summary of ICT in your organization, your software related activities and roles, your business model, and your software integration approach.
2. What is your typical process for making architectural decisions?
3. What are your challenges (technical and social) in making the decisions?
4. How do you identify architecturally significant requirements from a set of architectural concerns and business context?
5. Who are involved in the analysis process and how do they collaborate?
6. Do you search for alternative solutions for your requirements when you make decisions? Even if you already had a solution in mind?
7. How do you select among alternative solutions? Do you consider and reuse architectural patterns, styles, reference architectures, industrial standards, etc.?
8. Do you reuse the already made decisions between your various projects?
9. How do you validate your final solution? Do you use some approaches like ATAM, CBAM, etc.?
10. Do you validate your architectural solution only in design stage or even later when the whole system is launched?

As it is clear in the above list, the RQ3 is not covered by any of the questions. After doing the first interview, we realized that the effect of SECO relationships is an

important influencing factor on the decision-making process and we had not considered it. So we added it to the questionnaire for the next interviews. To this end, we added specific questions for each category to explore the subject from each category aspect. For example we were interested to see whether the decisions in the software vendors are made mainly by them or it is more customer-driven. The similar question was asked from the customers but from the opposite direction to see whether their decisions are affected by vendors or consultant companies.

After finalizing the questionnaire and making appointments with interviewees, the interviews were conducted. Since the data collection method was semi-structured interview, we considered some flexibility in asking the questions based on the answers we got from the interviewees. It means the above set of questions was more an interview guide; some questions would be skipped (for example if the interviewee didn't have any idea about the question) and some new questions would be added during the interview.

To conduct the interview with grid utilities, in some cases the interview had two parts: more general questions were answered by a project manager or head of ICT and the more detailed questions by a software architect or developer. So totally 8 interviews were conducted of which 4 were face-to-face and 4 were through Skype. Among 8 interviewees, 4 were heads of ICT, 2 were software architects, one was software developer and one was project manager. The interviews were captured by a voice recorder and lately were transcribed.

For analyzing the interview data, the step-by-step thematic synthesis proposed by Cruzes and Dybå was applied. It is mainly proposed for a systematic literature review, but is applicable for analyzing the qualitative interview transcripts similar to stage-by-stage method by (Burnard, 1991). The essential aim of these methods is to increase the abstraction level of transcribed texts from the text level to the code and theme level and create taxonomy of higher-order themes (Cruzes and Dybå, 2011). We did the same to reach from the interview results to the answer of our research questions.

4 Results

By thematic synthesis we extracted 18 codes from interview transcripts and those 18 codes were categorized into 4 themes that are described in the following sections. Section 4.1 and 4.2 correspond to RQ1, section 4.3 is related to RQ2 and section 4.4 refers to RQ3.

4.1 Making Architectural Decisions for Enterprise Application Development

In all of the energy companies we interviewed, there is an IT section either as a department (if the organization composes of only one company) or a company within

the whole group (if the organization consists of many companies). IT section has a general roadmap or high level strategy for making architectural decisions regarding software enterprise application integration. For instance in some grid utilities the rules from IT sections imply that new products should have proper interfaces or adaptors to be integrated through ESB (enterprise service bus). Or they are emphasizing on reducing information redundancy by engaging SOA-based development. Some of the grid utilities have developed their guidelines based on some international frameworks for developing enterprise architecture. Later on when every department wants to make architectural decisions for their projects, they should make their decisions in alignment with guidelines and principles from IT section of the organization.

The lower level architectural decisions are made at project level in different departments. So each department has either its own software architects or they hire architects from IT department. Then the decisions are made by several meeting among the software architects and project manager (or product owner). If there is a decision about a common solution like ESB, it is rather made at the IT department.

Most of the grid utilities are not familiar with the terms and concepts like ATAM. In practice they are conducting some structured procedure which could be more informal than approaches like ATAM but still they are satisfied with the results. They have several meetings among related stakeholders, define possible alternatives and look at their possible advantages and disadvantage, consider the important non-functional requirements, also look at their organizational limitations and project schedule, and select a solution among alternatives. Some of them do the proof of concept for the selected solution to check whether the solution supports their business requirements.

4.2 Using Standards for Making Architectural Decisions

Although there have been some standards for software integration in Smart Grid for several years (for example IEC 61970 which is also called CIM), almost none of the grid companies have done their integration based on those standards and this makes their architectural decisions more challenging. They are now becoming more aware of the need to apply standards to reduce the interoperability challenges, so they are exploring the standards and are going to use them. Also some of the software vendors are starting to deliver their products based on those standards. So the future of the Smart Grid in Norway would be standard based but currently is not.

4.3 Documenting and Reusing the Architectural Decisions

Some of the companies document important architectural decisions. To this end, they either keep the meeting minutes or use an internal wiki for documentation rather than a specific tool. One of the documentation issues they often have with these approaches is

the maintenance of the documentations, especially when it comes to the decisions about software integration, which these days is very dynamic in these companies. So it is hard for them to always update the latest version of the documents.
The situation of reusing the decisions depends on the level of decisions. In high level decisions, 4 out of 6 companies are reusing their high level software integration processes across different projects. The reuse happens in an ad-hoc manner and as it was explained earlier, it is mainly done by setting some high level rules or guidelines by IT departments and different projects should apply them in their integration. One of the companies is composed of both energy and telecommunication parts and is an interesting case in this aspect because overall processes which have been developed in telecommunication is reused in the smart grid initiatives too.
When it comes to the low level architectural decisions almost none of the companies are reusing the decisions across different projects and the decisions are not transferring between different projects in a written sense. The decisions are rather kept in the head of the decision makers even if they are participating in different projects. So some of the companies showed their interest to be familiar with reusable architectural decision frameworks and believe that it would be useful to learn from history and apply the experiences from previous projects in the future projects. One of the companies has a successful experience where for choosing ESB they have reused the requirements and available alternatives from other companies and they were satisfied with the results.

4.4 Effect of Software Ecosystem Relationships on the Architectural Decisions

Most of the grid utilities (customers) believe that the market is vendor driven while the software vendor believes that the market is customer driven. The examples from both sides confirm both claims. So it essentially means that as an actor of an ecosystem each of them affect the other one. From customers point of view they are limited to what vendors deliver and from the vendors side they limit their development to what the customer require. Besides, there are some regulating organizations that also affect the choices of software integration in grid utilities. So in general the interviews showed that the relationship among different actors of the software ecosystem affects the architectural decisions in each actor. The effect of these relationships is described as follows.

Effect of Regulators on the Architectural Decisions of Grid Utilities

One obvious example to show the effect of regulators on the architectural decisions in the grid utilities is SCADA (supervisory control and data acquisition) system. The grid utilities want to have a fully integrated system and therefore desire to add SCADA to their SOA-based integrated system as well, but the security regulations from Norwegian

electricity regulator, which in opinion of grid utilities are old-fashion, have limited them. So they should treat SCADA as a silo system and do all the interactions and information exchange manually.

Effect of Vendors on the Architectural Decisions of Grid Utilities

An interesting example that shows the effect of vendors on the architectural decisions is a situation in one of the grid utilities where they wanted to decide choosing between IPv6 and the lower versions. For launching AMS (advanced metering system) project, they will install more than 200,000 IP-based devices in their municipality, so technically they preferred IPv6. But their challenge was that most of current vendors don't deliver products that support IPv6. Now some more professional vendors like Cisco are joining the Smart Grid market and that grid utility has finally decided to use IPv6. Another example by the same grid utility is a decision about separating the database of DMS (distribution management system) from other systems. The reason for the decision also relates to what the vendors deliver. The DMS should use a NIS (network information system) and a NIS itself is based on a GIS (geographical information system). The problem is that the current GIS suppliers don't have a electric schematic layer. What is now on the GIS is a general network of nodes and edges. So the DMS the grid utility has bought should have a separate database to include electric schematic layer.

Effect of Customers on the Architectural Decisions of Vendors

The interviewed software vendor with an example showed how their architectural decisions depend on what the customers require. The vendor has two alternatives for deliver their products based on SOA: WS or REST-based services. Although they are aware of some advantages of REST-based services they are still stick to the WS and the reason is that none of their customers have asked for REST-based services in their request for proposal.

5 Discussion

In this part we discuss our findings in position with the previous findings from related literature. The part is organized based on our three research questions. In addition, section 5.4 discusses the possible threats to validity.

5.1 Architectural Decision-Making for Enterprise Application Development

As we discussed in the related work, the study by van Heesch and Avgeriou is a relevant empirical research about architectural decision-making in industrial companies

that is actually the aim of our RQ1 too. One of the results of their study is that the greatest part of architects doesn't follow particular architecture approaches from the literature (such as ATAM, SAAM, goal-oriented paradigm, etc.), they rather adopt architecture activities to define their own customized approach to making architectural decisions (van Heesch and Avgeriou, 2011). The result of our study also in line with their finding showed that most of the companies are not using systematic approaches such as ATAM to make and evaluate their architectural decisions. Even thought, it doesn't mean that the companies are making their architectural decisions in a totally intuitive way. Both our results and findings from van Heesch and Avgeriou's survey show that the companies identify architecturally significant requirements (architectural analysis), find different candidate solutions for the requirements consider advantages and disadvantages for candidate solutions (architectural synthesis) and validate the chosen solution against the requirements (architectural evaluation) (van Heesch and Avgeriou, 2011). In spite of similarities, different companies of our study have different procedures for each of mentioned processes. For example for architectural evaluation some use proof of concept while some launch real industrial prototype to evaluate the chosen solution.

5.2 Reusing Architectural Decisions

Zimmermann has done a significant work on reusing the architectural decisions for enterprise application development. Before that not many researches have been conducted on this topic (Zimmermann, 2009). One of our aims was to find out whether the interviewed enterprises predict the required architectural decisions in a new project based on experiences from previous projects that have been done in either their department of other departments of the same organization. As results show, some of them reuse high-level architectural decisions in term of architectural guidelines or rules but none of them reuse the low-level architectural decisions across various projects.
Reusable architectural decision model (RADM) developed by Zimmermann has been evaluated by several case studies and the results show how efficient it would be to reuse the architectural decisions in similar projects (Zimmermann, 2009). Zimmermann has employed his model in different industrial cases, from software vendors to software consultant companies and large-scale enterprises like telecommunication companies. But he has mainly applied his model on several projects within each company. What we observed through the interviews was the potential to also reuse the architectural decisions across different companies within a software ecosystem. Some of the companies have had collaboration on either writing requirement specification for an enterprise solution (e.g. ESB) or developing reference architecture for smart grid. Applying reusable architectural decisions frameworks like RADM would be very promising for these collaborations and through that the new requirements and

justifications to improve RADM would be gained.

5.3 Effect of Software Ecosystem Relationships on the Architectural Decisions

The results of our study show that the relationships among the actors of a software ecosystem could significantly affect the architectural-decision making process in each of the actors. Some previous studies have also discussed the non-technical influences on the architectural decisions. Van Heesch et al. have defined architectural decision forces as any aspect of an architectural problem arising in the system or its environment to be considered when choosing among the available decision alternatives (van Heesch et al., 2012). The non-technical forces they have talked about are personal preferences or experience of the architects, business goals such as quick-time-to-market, low price, or strategic orientations towards specific technologies (ibid). Their reference for considering the influence factors on software architecture is an empirical study by Mustapic et al. that have been conducted to investigate the possible real world influences of software architecture (Mustapic et al., 2004). What they have gained as the influence factors are relationships of system, computer hardware and software architecture, reuse and legacy in architectural design, business and application domain factors, choice of technologies, organizational factors, process related factors and resources used for architectural design (ibid). The most relevant factors to our results are business and application domain factors and organizational factors. The more specific factors they have investigated for these categories are standards, type of customers, production volume, product lifetime, non-functional requirements, distributed development, outsourcing, size and maturity of organization (ibid). So the SECO relationships have not been explicitly covered by the mentioned studies and the results of our study can be considered as a decision factor in addition to what they have extracted before.

5.4 Threats to Validity

Internal. One potential threat to internal validity of our research is that there were too few interviews to make reliable results. However, all of the companies were the largest enterprises and software vendor in the same software ecosystem. It means that there were few differentiations between the characteristics of the companies (type, size, products, business processes, structure, etc.). Also there is little disagreement among the interviewees from different companies. Therefore we do not assume that interviewing more companies will result to different conclusions. Even though, interviewing with software consultant companies and regulating organizations in the same software ecosystem would increase the reliability of the results. As we mentioned earlier, we couldn't convince them to participate in our study within the time schedule we had.

External. The generalization of the results to all large-scale enterprise based on the

enterprises from one industrial domain is arguable. Although large-scale enterprises from other domains like telecommunication, finance or health-care have also challenges on making architectural decisions for enterprise application development, our study shows that the software integration in the electricity industry is more immature than other areas due to lack of standardization and it can affect the architectural decision issues. So we are aware of the threat to external validity and conducting the same interviews with enterprises from other domains would make the results more reliable.

6 Conclusions and Future Work

In this paper we presented the result of interviews with six companies within software ecosystem of Norwegian electricity industry being five grid utilities and one software vendor. Our main goal was to empirically investigate the architectural decision making and reusing situation in the large-scale enterprises to enrich the state of practice in the architectural decisions area. We gained three explicit results:

1) In line with few previous empirical studies, our study show that most of the companies are not using well-known academic approaches such as ATAM, they are rather using their own structured procedures.
2) The relationships among the actors of a software ecosystem could significantly affect their architectural-decision making processes for example by limiting their alternative solutions. This factor should be also considered as an influencing factor on architectural decision making process in addition to the factors previous studies have extracted from the industry.
3) There is a high potential among enterprises to reuse the architectural decisions across their various projects or across different companies within a software ecosystem. The previous reusable architectural decision frameworks have been applied mainly in various projects within one company while our study shows that such frameworks can be applied also in different companies within a software ecosystem that have some kind of collaboration.

In the next step, we are going to apply reusable architectural decision frameworks (such as RADM by Zimmermann or decision forces viewpoint by van Heesch et al.) on some of the large-scale enterprises or software consultant companies in the Norwegian electricity industry. By doing such case studies we are going to investigate how these frameworks can be improved and customized for the electricity industry based on the feedbacks we get from the case studies.

Acknowledgements. The authors would like to thank all interviewees for their participation and their valuable responses.

7 References

1. Babar, M. A., Dingsøyr, T., Lago, P. and van Vliet, H.: Software Architecture Knowledge Management, Springer (2009).
2. Brinkkemper, S., Soest, I.V. and Jansen, S.: Modeling of Product Software Businesses: Investigation into Industry Product and Channel Typologies, in: C. Barry et al. (eds.), Information Systems Development: Challenges in Practice, Theory, and Education, Vol.1, pp 307-325 (2009).
3. Burnard, P.: A Method of Analysing Interview Transcripts in Qualitative Research, Nurse Education Today 11: 461-466 (1991).
4. Chen, D., Doumeingts, G. and Vernadat, F.: Architectures for Enterprise Integration and Interoperability: Past, Present and Future, Computers in Industry, 59 (2008): 647-659.
5. Cruzes, D. S. and Dybå, T.: Recommended Steps for Thematic Synthesis in Software Engineering, In the Proceedings of the 5th International Symposium on Empirical Software Engineering and Measurement, ESEM 2011, Banff, AB, Canada, (2011).
6. Falessi, D., Cantone, C., Kazman, R., and Kruchten, P.: Decision-Making Techniques for Software Architecture Design: a Comparative Survey, ACM Computing Surveys 43 (4) (2011).
7. Fisher, D. A.: An Emergent Perspective on Interoperation in Systems of Systems, Software Engineering Institute, Technical Report, CMU (2006).
8. Hoorn, J. F., Farenhorst, R., Lago, P. and van Vliet, H.: The Lonesome Architect, The Journal of Systems and Software 84 (2011), pp. 1424-1435.
9. Ivanovic, A. and America, P.: Information Needed for Architecture Decision Making, Proceedings of the 2010 ICSE Workshop on Product Line Approaches in Software Engineering, pp. 54-57 (2010).
10. Jansen, S., Finkelstein, A., and Brinkkemper, S.: Business Network Management as a Survival Strategy: A Tale of Two Software Ecosystems. In Proceedings of the First Workshop on Software Ecosystems. CEUR–WS, vol. 505 (2009).
11. Kruchten, P., Lago, P., van Vliet, H.: Building up and Reasoning about Architectural Knowledge. In: C. Hofmeister, I. Crnkovic, R. Reussner (eds.) Quality of Software Architectures, Proceedings 2nd International Conference, LNCS, vol. 4214, pp. 43–58. Springer, Berlin (2006).
12. Lucassen, G., Brinkkemper, S., Jansen, S. and Handoyo, E.: Comparison of Visual Business Modeling Techniques for Software Companies, Software Business (2012): pp. 79-93.
13. Maier, M. W.: Architecting Principles for Systems-of-Systems, Systems Engineering, 1, 4 (1998): 267–284.
14. Mustapic, G., Wall, A., Norstrom, C., Crnkovic, I., Sandstrom, K., Froberg, J. and

Andersson, J.: Real World Influences on Software Architecture – Interviews with Industrial System Experts, Proceedings of the Fourth Working IEEE/IFIP Conference on Software Architecture (WICSA) (2004).

15. Perry, D. E., Porter, A. A. and Votta, L. G.: Empirical Studies of Software Engineering: A Roadmap, Proceedings of the Conference on The Future of Software Engineering, p.345-355, Limerick, Ireland (2000).
16. Robson, C.: Real World Research: A Resource for Users of Social Research Methods in Applied Settings. 3rd ed. Chichester: Wiley; (2011)
17. Tang, A., Babar, M. A., Gorton, I. and Han, J.: A Survey of the Use and Documentation of Architecture Design Rationale, 5th Working IEEE/IFIP Conference on Software Architecture (WICSA), pp.89-98, (2005).
18. Tang, A., Avgeriou, P., Jansen, A., Capilla, R. and Babar, M. A.: A Comparative Study of Architecture Knowledge Management Tools. Journal of Systems and Software. 83, 3, pp.352-370 (2010).
19. van Heesch, U. and Avgeriou, P.: Mature Architecting – A Survey about the Reasoning Process of Professional Architects, 9th Working IEEE/IFIP Conference on Software Architecture (WICSA), pp.260-269 (2011).
20. van Heesch, U., Avgeriou, P. and Hilliard, R.: Forces on Architecture Decisions – A Viewpoint, Proceeding of Joint Working Conference on Software Architecture and 6th European Conference on Software Architecture, pp. 101-110 (2012).
21. Zimmermann, O., Koehler, J. and Leymann. F.: Architectural Decision Models as Micro-Methodology for Service-Oriented Analysis and Design. In SEMSOA Workshop, Hannover, Germany, (2007).
22. Zimmermann, O.: An Architectural Decision Modeling Framework for Service-Oriented Architecture Design. PhD Dissertation, University of Stuttgart (2009).

P2- Towards Reusing Architectural Knowledge as Design Guides

Published: In Proc. The 26th International Conference on Software Engineering and Knowledge Engineering, SEKE 2014.

Towards Reusing Architectural Knowledge as Design Guides

Functional Requirements, Tool Analysis and Research Roadmap

Mohsen Anvaari[1], Olaf Zimmermann[2]

[1]Norwegian University of Science and Technology, Trondheim, Norway
mohsena@idi.ntnu.no

[2]University of Applied Sciences of Eastern Switzerland, Rapperswil, Switzerland
ozimmerm@hsr.ch

Abstract. In recent years, architectural knowledge management has demonstrated its potential to improve software development and evolution practices; various tools and research prototypes now exist for documenting architectural knowledge. However, capturing such knowledge is not enough: according to practitioners' feedback, a certain amount of knowledge post-processing is required to make the captured knowledge consumable and stimulate reuse. In our previous work, we created a method for enhancing knowledge about the past (decisions made) into architectural guidance for the future (decisions required). However, additional concepts are required to let our method benefits from recent advances in architectural knowledge management tool engineering. In this paper we establish requirements for post-processing architectural knowledge captured on projects and enhancing the knowledge into architectural guidance. The requirements are derived from literature and industrial experiences. Next, we analyze existing tools with respect to these requirements. Finally, we establish a vision for an integrated method and tooling for architectural guidance modeling and outline a roadmap for future research and tool development towards this vision.

Keywords. Architectural knowledge; decision reuse; architectural synthesis; design guide; knowledge management tool

1. INTRODUCTION

Architectural decisions are considered a first class entity in software engineering now 0; researchers define software architecture as a set of architectural design decisions 0. Various tools and research prototypes exist (or are under development) for documenting the architectural knowledge. Although most practitioners are still reluctant to use formal templates and tools that academic researchers have developed 0, our observations show that many organizations have started to capture their architectural decisions 0. This is often done in light and pragmatic ways, e.g., using simple wikis or chronological meeting minutes 0.

According to studies on inhibitors for knowledge reuse, documenting the knowledge is not enough; post-processing is required to stimulate the reuse and make the knowledge consumable 0. Hence, we created a method for enhancing knowledge about the past (decisions made) into architectural guidance for the future (decisions required) 0. However, additional concepts are required to let our method benefits from recent advances in architectural knowledge management tool engineering. Therefore, the goals of this paper are:

1. To specify the requirements for tools that facilitate the post-processing of captured architectural knowledge from projects and enhancing such raw knowledge into design guides for future decision making activities.
2. To analyze existing tools and research prototypes with respect to the proposed requirements.
3. To establish a vision for an integrated method and tooling for architectural guidance modeling.

The rest of the paper is organized as follows: In Section 2, we present related work. Section 3 describes our research method. Section 4 specifies the functional requirements for tools that support enhancing architectural raw knowledge into reusable design guided. These requirements are derived from the authors' industrial experience as well as a review of research prototypes and tools. Section 5 reports on the results of our analysis of existing tools and research prototypes with respect to the functional requirements from Section 4. Section 6 analyzes our results and establishes an architectural vision for a tool that supports architectural knowledge reuse. Section 7 summarizes the paper with conclusions.

2. RELATED WORK

The concept of architectural knowledge – defined as the integrated representation of the software architecture, the architectural decisions, and the external context/environment 0 – has been investigated by researchers since they started to consider the architectural decisions as important entities of a software system just like the architecture itself. In the last decade, the research community has elaborated the concept, clarified its definitions, terminologies and boundaries, established the ways of presenting the knowledge, and developed the approaches to manage the knowledge 0.

Applying general knowledge management principles 0 to the software engineering domain, two activities become essential for architectural knowledge management: creating (or capturing or documenting) knowledge and consuming (or reusing or applying) knowledge. However, as we have shown in our previous work, the main focus of the software architecture community so far has been on knowledge capturing, not on knowledge reuse 0. One may argue that when the knowledge is captured and made available to the others it is reusable; therefore any approach and tool that supports knowledge capturing also supports knowledge sharing and reusing automatically. But, according to practitioners' feedback, capturing the knowledge is not enough: a certain amount of knowledge post-processing is required to make the captured knowledge consumable 0. Examples of knowledge post-processing are anonymizing the knowledge (e.g., remove sensitive personal information such as names of actual people, or replace them with role definitions such as "application architect" or "integration architect"), connecting the related knowledge entities (e.g., a decision about a message exchange pattern with a decision about a messaging provider software product), and removing the project-specific knowledge (chosen alternative) to make the knowledge reusable for other projects.

While the method we have created in our previous work for reusing architectural knowledge and enhancing the captured architectural knowledge into design guides[1] has demonstrated to be useful in the industry 0, better tool support is required to make the application of the method more efficient. As the first step, this paper explores the available architectural knowledge management tools and research prototypes to analyze how much they provide the required functionalities for architectural knowledge reuse and guidance development.

This paper is not the first survey in the software architecture domain to analyze and evaluate the architectural knowledge management tools. At least four preceding research papers have been published 0000. Although some papers have considered knowledge sharing as a functional requirement in their evaluation framework, their focus is not on knowledge post-processing and enhancing the knowledge into design guidance, which is the focus of our work. Furthermore, the last survey has been published in 2010 while in the last three years more tools and research prototypes have been developed or are under development. This paper covers these new tools as well. The mentioned research papers are valuable for our work; for instance, we have reused some of their functional requirements to establish the functional requirements in the section 4. In the next section, the method of the research will be explained.

3. RESEARCH METHOD

As Fig. 1 shows, we started the research by creating functional requirements. To do so we explored three sources: 1) industrial experiences that originate in the authors' contribution in industrial projects and also their observations from various industrial domains (software development projects for Smart Grid, financial applications, etc.) 2) the tools and research prototypes that are not accessible but are specified in the literature 3) the tools that are accessible publicly on the internet. The main way to reach to the second source was exploring the literature that has reviewed and compared the architectural knowledge management tools. As we mentioned earlier, the last comparative study of the architectural knowledge management tools was conducted in 2010 [2]. It still is a valuable source to explore the tools that had been developed until then. We discovered the newer tools either through the literature that tool developers have published or by contacting the researchers that we were informed are developing a tool.

The second step of the research was to analyze the tools and research prototypes with respect to the functional requirements we proposed in the first step. The tools that are publicly accessible (the third source) were a more valuable source for us, because we could actually use them and test their functionalities against the list of requirements.

[1] A design guide is a reusable asset containing knowledge about architectural decisions required in a particular domain 0. As a reusable asset 0, a design guide has been curated, edited and quality assured for readability and reuse. We refer to this curation and editing as knowledge post-processing.

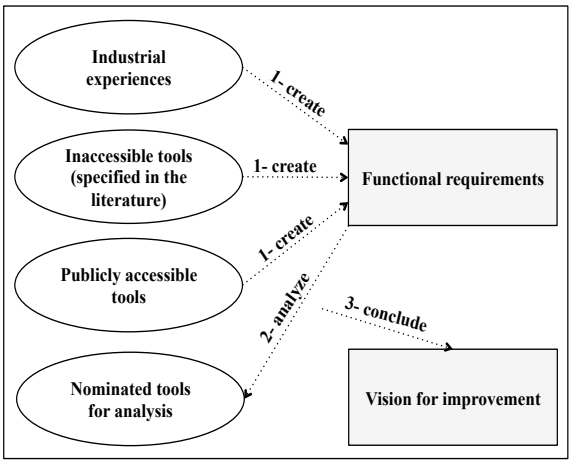


Figure. 1 Research activities and contributions

Some of the inaccessible tools were also valid for analysis since their functionalities have been described concretely in the literature. We finalized the list of tools for analysis using the following criteria:

- The tool should be publicly accessible. For example, ADkwik 0 that is covered by previous tool evaluation studies 0 would be a candidate for our analysis, but it is not accessible anymore. However, there are some tools that are under development and therefore are not released yet, but their functional requirements are concretely described in literature; such tools do meet the criterion. One such tool is analyzed in Section 5.
- The installation and usage of the tools should be straightforward.
- The tool should be representative for its domain. For example if tool A and B exist for capturing the decisions, and tool B covers all features of tool A, we just choose tool B.

The third and last step was to summarize the tool analysis results. Based on the analysis results, we establish a vision for developing a tool that supports post-processing the captured architectural knowledge and developing a design guide.

4. FUNCTIONAL REQUIREMENTS

This section describes functional requirements for developing an architecture guidance modeling tool. We will use the functional requirements to analyze the existing tools later (in Section 5). In the following, we will define some terms that are essential for describing the functional requirements first. Next, we will describe the actual functional requirements.

A. Definitions

The required tool for supporting architecture guidance modeling should create and maintain a knowledge base (KB) that contains architectural entities. The tool should be able to post-process the entities (see Section 2 for examples) and enhance them into a design guide. Inspired by our previous work 0, we define the following entities that can be added to a KB:

- Issue: Any design issue that may occur in a software development project. It includes different properties mainly name, problem description, decision drivers and solution alternatives. Each alternative includes pros and cons, known uses and related background. For example in an enterprise architecture, an issue can be "enterprise integration pattern for designing the message channel between system A and system B". The alternatives of the issue are "point-to-point-channel" and "publish-subscribe-channel".

- Decision: A decision inherits its properties from an issue, but adds the outcome of the decision (chosen alternative and the rationale behind that). Therefore an issue can be converted to a decision by adding the outcome and vice versa. In the example we provided, the outcome may include "the publish-subscribe-channel" as the chosen alternative and "high number and change rate of the data sinks" as the rationale of the decision.

- Group: A group entity is an aggregation (or assembly) of other entities. An example for the usage of this container concept is a software project that includes some sub-projects and each sub-project includes issues and decisions.

This structure is not the only way of modeling the architectural entities. Tools can apply other metamodels such as those presented in 0 or 0.

B. Functional Requirements

We categorize the functional requirements in two groups: 1) create and maintain knowledge and 2) consume knowledge. However, these categories have some overlaps and some requirements can belong to more than one category.

1) Create and Maintain Knowledge

- AddE – Add an entity 000: Insert an entity to KB. The following features are required:

  - *Rich text editor*
  - *A tag field (or a semantic-based approach) to make search easier*
  - *Entity identification*
  - *Entity name*
  - *Entity description*
  - *Entity stakeholders*
  - *Entity version*
  - *Entity confidentiality*

- *Issue level* (e.g. conceptual, technology, vendor asset 0)

Before the tool inserts an entity to KB, it should first search for available related entities and if there are some, it should suggest them to the user. If the user finds that the entity is already available in KB, (s)he can decide to cancel the procedure. This helps to reduce the redundancy. Sometimes the entity is not already available, but the search brings some related entities and the user can connect the new entity to the related ones (CnctE).

- UpdE – Update an entity 00: Update an available entity. The features that are required for AddE apply here as well.

- RmvE – Remove an entity 0: Remove an entity from KB. It should clean up all of the relations of the entity to the other entities.

- MovE – Move an entity 0: Move an entity from one group to another group.

- CnctE – Connect (Relate) to an entity: Connect an entity to other entities (examples are issue to issue, issue to decision, issue to group).

- UlnkE – Unlink an entity: Unlink an entity from its parent or from its related entities without deleting the entity.

- RevE – Review an entity 00: If an entity is supposed to be reviewed before inserting to KB, it should be sent to the reviewers. The reviewers should validate the entity before a specific time. Then based on the rates or opinions the reviewers give to the entity, the entity can be rejected, inserted (or edited and inserted) to KB.

- ImpE – Import an entity 0: Import an entity from a file (for example a XML or JSON file) or a URL.

- AnmE – Anonymize an entity 0: Sometimes an entity can be reused or shared or exported, but the project-specific information should not be shared with the others. This feature replaces the project-specific terms with a pseudonym.

- MkeD – Make a decision (convert an issue to a decision) 0: When a decision on an issue is made, the decision should be inserted to KB. This functionality adds an outcome part to the issue and converts it into a decision.

- GnrI – Generate an issue from a decision: Sometimes a decision is made, but the related issue is not in KB. This functionality generates an issue from a decision by removing the outcome part of the decision entity (and adding a possible recommendation). This feature is essential for upgrading decisions to guides (UpgG).

- UplD – Upload documents 0: An entity may include background information either as a link or as a document. This feature uploads a document to the entity.

- NtfS – Notify stakeholders 0: Sometimes an entity would have more than one owner (stakeholder). This feature reports any changes to the entity to all stakeholders. Stakeholders should be able to disable/enable this feature. The implementation examples of this requirement are RSS (Rich Site Summary) and Atom 0.

• Conf – Configure the tool: The metamodel (default profile) that are required to insert an entity (such as entity version, entity stakeholders, entity confidentiality, issue level and so on; check AddE for more details), should not be fixed and unchangeable. This functionality customizes the attributes based on the project or organization needs.

2) Consume Knowledge

• SrhE – Search an entity 00: Search KB for a group, an issue or a decision. The search can be done based on an entered text or by choosing an entity to find the relevant entities. Advanced search to limit the search results based on level, confidentiality, project, etc. should be supported.

• LstE – List entities: Make a list of all entities of a group.

• NKB – Navigate the knowledge base 0: The user should be able to navigate between the groups, their sub-groups and their issues and decisions.

• ViwE – View an entity 000: When the user finds the list of entities by searching them, or navigating KB, (s)he should be able to view each entity and its properties.

• RuseE – Reuse an entity 00: Choose an entity in group A and copy it into group B. The confidentiality (access permission) of the entity should be checked first. The owner of group B might be able to view the entity of group A, but not to reuse it.

• ExpE – Export an entity 0: Export an entity to a file. The confidentiality of the entity should be checked first: if export is not allowed by the owner of the entity, the export procedure should be rejected. If anonymized export is allowed, the entity should be anonymized first and then be exported. If export is allowed unconditionally, the entity can be exported without any pre-processing.

• ShrE 00 – Share an entity: Send the link of an entity to other stakeholders by email notification. First the confidentiality of the entity should be checked. Sharing is possible only if the entity is public or other stakeholders have access to the group.

• UpgG – Upgrade to guide: Convert past architectural knowledge into guidance for the future. First, the tool will ask about anonymization. If the user requests anonymization, the feature will anonymize all entities of the group (AnmE), otherwise leave them unchanged. The next step is to look up each decision and its related issue. If both a decision and its related issue are available, the feature will remove the decision (RmvE). If only the decision is available, it will generate an issue from the decision (GnrI) and remove the decision (RmvE) afterwards. The final result is a group and all of its sub-groups and each sub-group includes a list of design issues and each issue has some alternatives. This can be shared (ShrE) or exported (ExpE) as a design guide. Assume that a software developing firm has a project for developing a system for customer A. They have captured the decisions in a group called project A. The group includes various sub-groups (A1, A2, A3, etc.). Now by using this feature, they will have a list of issues and alternatives for each of the sub-groups and they can use it as a guide for making decisions in a similar project for developing a system for customer B. The architects and

designers involved in the new project could be different, but the knowledge from previous project is reused in a structured manner.

• DAPI – Documented application programming interface: The tool should provide a public documented API to make it possible to be integrated with other architectural knowledge management or design modeling tools.

In the next section, we present the results of analysis of existing tools with respect to the mentioned functional requirements.

5. ANALYSIS RESULTS

This section reports on the results of our analysis of existing tools and research prototypes with respect to the functional requirements from Section 4. First, we briefly introduce the five tools that are nominated for the analysis, and then we present the functional requirements that are satisfied by these tools.

1) *SAW.* Software Architecture Warehouse (SAW) is a Web-based tool to capture, manage and analyze architectural knowledge. It is implemented to help the entire software architecture design team achieve situational awareness about architectural decisions 0.
2) *Decision Viewpoints.* Decision Viewpoints is a documentation framework for architecture decisions. It uses the conventions of ISO/IEC/IEEE 42010 0. A tool is developed supporting the framework as an add-in for Sparx Systems' Enterprise Architect 0.
3) *AREL.* Architecture Rationale and Elements Linkage (AREL) is a Sparx Systems' Enterprise Architect plug-in that creates architectural design with a focus on design rationale 0.
4) *SEURAT.* Software Engineering Using RATionale system (SEURAT) is an Eclipse plug-in that aims to manage architectural knowledge from requirements to source code 0.
5) *Eclipse Process Framework (EPF)*[2]. EPF is an Eclipse-based method creation tool. In EPF, knowledge creation takes place in the tool; knowledge consumption, on the other hand, can be done in a Web browser.

Table I. shows which functional requirements are supported and which are not supported or partially supported by the introduced tools.

[2] http://projects.eclipse.org/projects/technology.epf

TABLE I. ANALYSIS OF TOOLS IN A NUTSHELL[3]

FR	SAW	Decision VP	AREL	SEURAT	EPF
AddE	Partially	Partially	Partially	Partially	Partially
UpdE	Yes	Yes	Yes	Yes	Yes
RemE	Yes	Yes	Yes	Yes	Yes
MovE	No	Yes	No	Yes	Yes
UlnkE	Yes	Yes	Partially	No	Yes
CnctE	Yes	Yes	Yes	No	Yes
RevE	No	No	No	No	Partially
ImpE	Partially	Partially	No	No	Yes
AnmE	No	No	No	No	No
MkeD	Yes	Yes	Yes	Yes	No
GenI	No	No	No	No	No
UplD	No	Yes	Yes	No	Yes
NtfS	Partially	No	No	No	No
Conf	No	No	No	Partially	No
SrhE	No	Yes	Yes	No	Yes
LstE	Yes	No	No	No	Yes
NKB	Yes	Yes	Yes	Yes	Yes
ViwE	Yes	Yes	Yes	Yes	Yes
RuseE	Partially	Yes	Yes	No	Yes
ExpE	Partially	Partially	No	No	Yes
ShrE	Partially	No	No	No	Yes
UpgG	Partially	No	No	No	No
DAPI	No	No	No	No	Yes

6. VISION FOR FUTURE RESEARCH

In the previous section, we presented the results of our analysis. It showed the functional requirements that are supported by the available tools and research prototypes. Based on the data Table I. provides (the functionalities that are not focused by the available tools), we establish some directions for the next steps towards tool developing for architectural knowledge reuse and architecture guidance modeling:

1) *Separating issue (decision required) from decision (decision made):* A design guide is mainly a list of design issues (decisions required) and their possible solutions (alternatives). To create a guide from captured decisions (decisions made), issue and

[3] Detailed evaluation results omitted due to space constraints, but are available upon request.

decision should be separated and the tool should provide the possibility to generate an issue from a decision (GnrI). This will also make organizations less reluctant to share their knowledge with a community; because it guarantees that only the issue and its alternatives will be shared with the community and others will not be informed about their decision (chosen alternative).

2) *Providing default profile:* One of the main reasons architects state for their unwillingness to use architectural capturing tools is the time limitations 0. To overcome this, tools should make capturing knowledge less time consuming. One of the solutions is providing a default profile for adding entities to the knowledge base.

3) *Providing knowledge confidentiality:* As we mentioned earlier, organizations are not eager to share all of their architectural knowledge with the community. There can be even a situation that in one organization, the knowledge of one project should not be shared with other projects. Therefore the confidentiality level of an entity should be defined for adding the entity to the knowledge base. The tool should always consider the confidentiality and intellectual property rights level of an entity before sharing, reusing or exporting the entity or creating architecture guidance (e.g., "open", "copyright protected", "company-internal", and "confidential").

4) *Configuring metadata:* Users should be able to customize the metadata (attributes profile) based on their organizational policies and concerns. IEC/IEEE/ISO 42010 is one, but not the only template to be supported (many more have been defined, e.g. 0).

5) *Considering semantic tags:* The architectural knowledge base can grow very fast. Navigating a large knowledge base can be painful, e.g. if it takes a long time to find a knowledge entity. Providing semantic tags will make searching the knowledge base easier and more precise.

6) *Searching the knowledge base before inserting new knowledge:* To create a useful yet concise architecture guidance it is essential to reduce the redundancy of knowledge. To reach that, the tool should search the knowledge base in advance to inserting any new knowledge. It is also useful for finding relevant knowledge and connecting them together.

7) *Being consistent with real world situation:* In reality, large organizations develop software within various projects and sub-projects. The design guide would be more usable if it was categorized into projects and sub-projects. Therefore grouping the entities of knowledge base to projects and sub-projects should be provided.

8) *Anonymizing the knowledge:* Rather than separating issues from decisions, anonymizing the knowledge also makes organizations more eager to share their knowledge with the community (see Section 2 for an example of a required anonymization).

9) *Providing programming interface:* The activities related to architectural knowledge management are very wide and it is not possible to have a holistic tool that supports all activities. The focus of the proposed tool in this research is on reusing architectural knowledge and enhancing a design guide. The tool should therefore provide an interface

(API) to make it possible to be integrated with other architectural knowledge management or design modeling tools such as general-purpose wiki engines and Unified Modeling Language (UML) tools.

7. CONCLUSIONS

Reusing architectural decisions as design guides gives these decisions a more proactive role and therefore makes decision management more appealing and relevant to practitioners. In this paper, we leveraged our industrial experiences, our previous research work and also the current literature in the architectural knowledge community to establish functional requirements for future knowledge management tools that enhance architectural decisions to design guides. With respect to the functional requirements, we analyze representative tools and research prototypes. We reported that the available tools and research prototypes have made significant contributions in the area of architectural knowledge capturing, but still require a number of extensions so that the captured decision can serve as design guides in practice. We finalized the paper with a vision for method integration and tool improvement.

In the next step, we are going to evolve our design guidance enhancing framework to decrease the time and effort of design guidance generating by applying automatic information extraction approaches. The extracted architectural entities will feed the knowledge base (KB) in a more efficient way. We also intend to extend and integrate our method into existing and emerging tools (our own tools and those developed in the research community) – applying the vision we established in this paper.

REFERENCES

[1] A. Jansen, and J. Bosch, “Software architecture as a set of architectural design decisions”, WICSA 2005, pp. 109–120, 2005.

[2] A. Tang, P. Avgeriou, A. Jansen, R. Capilla, and M. A. Babar, “A comparative study of architecture knowledge management tools”, JSS 83(3), pp. 352–370, 2010.

[3] A. Tang, Y. Jin, and J. Han, “A rationale-based architecture model for design traceability and reasoning”, JSS 80(6), pp. 918–934, 2007.

[4] B. Hammersley, “Developing feeds with RSS and Atom", O'Reilly Media, Inc.", 2005.

[5] C. Manteuffel, D. Tofan, H. Koziolek, T. Goldschmidt, and P. Avgeriou. “Industrial implementation of a documentation framework for architectural decisions”. In Proc. 11th Working IEEE/IFIP Conference on Software Architecture (WICSA'14), pp. 225-234. IEEE, April 2014.

[6] H. C. Tan, P. M. Carrillo, C. J. Anumba, M. Asce, N. D. Bouchlaghem, J. M. Kamara, and C. E. Udeaja, “Development of a methodology for live capture and reuse of project knowledge in construction”, Journal of Management in Engineering 23(1), pp. 18–26, 2007.

[7] ISO, “ISO/IEC 42010: Systems and software engineering - Recommended practice for architectural description of software-intensive systems”, 2007.

[8] J. Burge and D. Brown, “SEURAT: Integrated rationale management”, ICSE 2008, pp. 835–838. ACM, 2008.

[9] J. F. Hoorn, R. Farenhorst, P. Lago, and H. van Vliet, “The lonesome architect”, JSS 84(9), pp. 1424–1435, 2011.

[10] J. Tvree. and A. Akerman, “Architecture decisions: demystifying architecture”, IEEE Software 22(2), pp. 19-27, 2005.

[11] L. Aggestam, and P. Backlund, “Strategic knowledge management issues when designing knowledge repositories”, ECIS 2007, pp. 528–539, 2007.

[12] M. A. Babar, T. Dingsøyr, P. Lago and H. van Vliet, “Software architecture knowledge management: theory and practice”, Springer, 2009.

[13] M. Anvaari, R. Conradi, and L. Jaccheri, “Architectural decision-making in enterprises : preliminary findings from an exploratory study in Norwegian electricity industry”, ECSA 2013, pp. 162–175, 2013.

[14] M. Biehl, “Literature study on design rationale and design decision documentation for architecture descriptions”, 2010.

[15] M. Nowak and C. Pautasso, “Team situational awareness and architectural decision making with the software architecture warehouse”, ECSA 2013, 2013.

[16] M. Shahin, P. Liang, and M. R. Khayyambashi, “Architectural design decision: existing models and tools”, 2009 Joint WICSA/ECSA, pp. 293–296, 2009.

[17] N. Schuster, “ADkwik – A collaborative system for architectural decision modeling and decision process support based on Web 2.0 technologies”, Stuttgart Media University, 2007.

[18] N. Schuster, O. Zimmermann, C. Pautasso, “ADkwik: Web 2.0 Collaboration System for Architectural Decision Engineering”, Proceedings of the Nineteenth International Conference on Software Engineering & Knowledge Engineering (SEKE 2007), Knowledge Systems Institute Graduate School, 2007. Pages 255-260.

[19] O. Zimmermann, “Architectural decisions as reusable design assets”, IEEE Software 28(1), pp. 64-69, 2011.

[20] O. Zimmermann, T. Gschwind, J. Küster, F. Leymann, and N. Schuster, “Reusable architectural decision models for enterprise application development”, QoSA 2007, pp. 157-166, 2007.

[21] O. Zimmermann, U. Zdun, T. Gschwind, and F. Leymann, “Combining pattern languages and architectural decision models into a comprehensive and comprehensible design method”, WICSA 2008, pp. 157-166, 2008.

[22] Object Management Group, “Reusable Asset Specification (RAS) Version 2.2.”, Available online at: http://www.omg.org/spec/RAS/2.2, 2005.

[23] P. Avgeriou, P. Kruchten, P. Lago, P. Grisham and D. Perry, "Architectural knowledge and rationale: issues, trends, challenges." ACM SIGSOFT Software Engineering Notes 32.4, pp. 41-46. 2007.

[24] P. Kruchten, “An ontology of architectural design decisions in software intensive systems”, In 2nd Groningen Workshop on Software Variability, pp. 54-61, 2004.

[25] P. Liang, and P. Avgeriou, “Tools and technologies for architecture knowledge management. (M. Ali Babar, T. Dingsøyr, P. Lago, & H. van Vliet, Eds.), pp. 91–111, 2009.

[26] U. van Heesch, P. Avgeriou, and R. Hilliard, “A documentation framework for architecture decisions”, JSS 85(4): pp. 795-820. 2012.

[27] U. van Heesch, P. Avgeriou, and R. Hilliard, (2012). “Forces on architecture decisions: a viewpoint”, 2012 Joint WICSA/ECSA, pp. 101-110, 2012.

P3- Semi-automated Design Guidance Enhancer (SADGE)

Published: In Proc. The 8th European Conference on Software Architecture, ECSA 2014.

Semi-Automated Design Guidance Enhancer (SADGE): A Framework for Architectural Guidance Development

Mohsen Anvaari[1], Olaf Zimmermann[2]

[1]Norwegian University of Science and Technology, Trondheim, Norway
mohsena@idi.ntnu.no

[2]University of Applied Sciences of Eastern Switzerland, Rapperswil, Switzerland
ozimmerm@hsr.ch

Abstract. Architectural decision making is a non-trivial task for architects in the software development projects. Researchers have developed several concepts, methods and tools to assist practitioners in their decision making and decision capturing activities. One of these approaches is a decision identification technique that creates architectural guidance models from decisions made in previous projects and from knowledge about a domain found in the literature. To apply this technique, significant manual knowledge engineering effort has to be invested initially. In this paper, we introduce a framework that automatically extracts architectural knowledge entities from architectural related documents by applying natural language processing. A knowledge engineer then manually post processes and fine-tunes the extracted knowledge entities. We applied evaluation techniques from the information retrieval community to measure the sensitivity and accuracy of the framework. Our results show that the automatic approach has the highest recall and shortest processing time while the semi-automatic approach has the highest precision.

Keywords: Architectural decision making, design guidance, information extraction, natural language processing, automatic annotation

1 Introduction

Architectural decision making is a non-trivial task for architects in software development projects. Since 2000, researchers have developed several concepts, methods, frameworks and tools to assist practitioners in their decision making and decision capturing procedures [2][6][11]. However, a recent study shows practitioners still have difficulties to make and manage decisions [13].

One of the difficulties the practitioners have in making decisions is recognizing and highlighting architectural issues in a specific project to make decisions about them (we call these issues *architectural issues* or *decisions required*). Our previous study shows that architects mostly rely on their intuitions to recognize architectural issues [1]. One of the promising approaches to help practitioners in their decision making is a decision identification technique that enhances architectural guidance (decisions required) from decisions made in previous projects and from knowledge about a domain that can be found in the literature. Through decision identification rules, this approach tasks a knowledge engineer to study pattern languages, genre- and style-specific extensions to software engineering methods, technical papers, industrial standards and project documentation to identify architectural issues [15]. To do so, the technique advises knowledge engineers to read the natural language texts of the documents and to annotate the texts manually. The intention is to extract architectural knowledge entities

(i.e., issues, alternatives, outcomes[1]) from documents and to develop an architectural guidance model from the extracted information. Such architectural guidance model is a reusable asset containing knowledge about architectural decisions recurring in a particular domain [16]. Several case studies have shown that the developed architectural guidance is promising in assisting the practitioners in their decision making, e.g. in SOA design and cloud computing [15]. However, this decision identification approach is manual; significant knowledge engineering effort that has to be invested initially before benefits can be realized.

In this paper, we introduce a framework called Semi Automated Design Guidance Enhancer (SADGE) that automatically extracts architectural issues (decisions required) from architecture documents by applying natural language processing (NLP) first (we refer to this automated step as automatic approach). In a second step, a knowledge engineer manually post processes and fine-tunes the extracted knowledge entities to increase the accuracy of the framework (we refer to the first automated and the second manual step together as semi-automatic approach). We validated and evaluated the SADGE framework in an experiment with students. The intention of this evaluation was to compare the effort, sensitivity and accuracy of architectural entities extraction process between manual, automatic and semi-automatic approaches. More specifically, by conducting the experiment we were going to find out:

- Research Question (RQ) 1: Which approach does have the shortest processing time for extracting the architectural entities?
- RQ 2: Which approach does have the highest sensitivity in extracting the architectural entities?
- RQ 3: Which approach does have the highest accuracy in extracting the architectural entities?

The contribution of this paper is twofold: 1) It applies NLP-based knowledge extraction to the architectural knowledge area and proposes a novel framework architecture and process model for doing so. 2) It demonstrates the efficiency, sensitivity and accuracy of this framework in enhancing architectural guidance from architectural related documents.

The rest of the paper is organized as follows. In the Section 2 we introduce the framework by describing how we have developed the framework and how users should operate and maintain the framework. Section 3 presents the design of the experiment and analyses and discusses the result of the experiment. Section 4 describes the related work in the software architecture domain. Finally, Section 5 concludes the paper and outlines future work.

[1] *Architectural issue* represents any design concern or problem that a decision should be made about; *alternative* presents a solution to the problem and *outcome* is the chosen solution among different alternatives [14].

2 SADGE – Framework for Semi-Automatic Architectural Knowledge Extraction

In this section, we explain how we developed the framework and how the framework operates.

2.1 SADGE Framework Development

SADGE has to be set up first (by a researcher) before practitioners in the projects can use it. As we mentioned earlier, the framework applies natural language processing (NLP) to extract architectural knowledge entities from a document. There are two main approaches in NLP to do so, machine learning approach and rule-based approach [5]. We tried both approaches, but due to the lack of enough training data, the machine learning approach did not work well; therefore in the current version of the framework we only use the rule-based approach. The stages of framework development will be described in the following subsections.

D1. Initializing the annotation rule. For developing the annotation rules we started with the simplest rule that an expert in the software architecture intuitively applies to manually annotate a sentence: *If a sentence contains at least one of the terms from catalog of terms* (a list of predefined keywords) *annotate it as an architectural issue.* For example, an architect would annotate the sentence "determine your validation strategy" in a document as an architectural issue (decision required) because of keywords "determine" and "strategy". Therefore, to apply the rule, the keywords (i.e., the catalog of terms) should be developed as well.

D2. Initializing the catalog of terms. To develop the catalog, we first interviewed an expert in the software architecture domain and identified terms that the expert considers as indicator to annotate a sentence as an architectural issue. When the first versions of the rule and the catalog of terms became ready, we applied them on some sample texts. We started by automatically annotating one document. To evaluate the result, we compared the automatic annotated entities against the entities that had been annotated manually by the expert. The evaluator presents the precision, recall, f-measure[2] and also shows those entities that have positively or negatively annotated.

D3. Evolving the annotation rule. When we applied the first version of the rule, the average recall of automatic annotation was high but the precision was very low. Hence, we decided to change the rule to reduce the amount of negatively annotated sentences and increase the precision. We divided the catalog of terms into two catalogs: high priority terms and low priority terms. Then the new rule is presented in Fig. 1.

[2] *Precision, Recall and F-measure* are the measures used in the information retrieval domain to measure how well an information retrieval system retrieves the relevant entities requested by a user. See [12] for definitions.

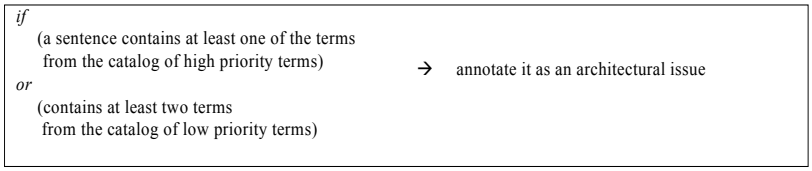


Fig. 1. The rule for annotating "architectural issues"

This rule resulted in a higher f-measure, so we replaced the first rule with the newer version. To decide whether a term is a low priority term or high priority term, we put the term in either category to see which one results to a higher f-measure.

D4. Evolving the catalog of terms. By looking at the sentences that should be annotated (according to the manually annotated text) but had not been annotated by the automated annotator, we found new terms to add to the catalog of terms. This resulted to higher f-measure. We added other sample texts one by one and did the same procedure for each text to develop the catalog of terms further. We finished the iterative procedure when the improvement of f-measure was not significant anymore. In total, we annotated seven documents that contained architecture related text. We selected the sample texts from various types of documents to make them representative in the software architecture domain. The texts were two industrial standards for software integration, three software design guidelines and two academic papers.

D5. Refining the catalog of terms. There is a possibility that some of the terms have positive impact on annotating one document whereas have negative effect on annotating some other documents. So in this stage of the framework development we decided to measure the impact of each of the terms on the average f-measure of all of sample documents. To do so, we removed each term from the catalog of terms and calculate the f-measure and then put the term back to check how the presence of a term affects the average f-measure. Those terms that their presence had negative effect on the average of f-measure were removed from the catalog of terms permanently. We call this stage of the development *sensitivity test*. The final version of catalog of terms after conducting the sensitivity test is presented in Fig. 2. We should mention that we use *stemming* for applying the annotation rule. Stemming is the process of reducing a word to its root. Therefore the terms in Fig. 2 are the roots of the terms and in the case another form of the word appears in a sentence the automated annotator considers it as an instance of the term. When both annotation rules and catalog of terms are developed, the framework is ready to be used.

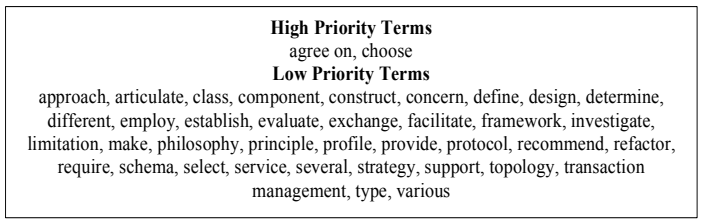


Fig. 2. Catalog of terms for annotating "architectural issues"

2.2 SADGE Framework Operation and Maintenance

The steps of framework operation and maintenance are described as follows. Note that in sections 1 and 3 by automatic approach we mean step O2 of the framework while O2 and O3 together make the semi-automatic part of the framework.

O1. Preparing documents for annotation. The input of the framework comprises text files that can be either project documents or domain literature. The knowledge engineer edits the text files in a way that the file doesn't include non-text objects (for example images) or non-relevant texts (cover page, table of contents, etc.). Then (s)he converts the text files to the types that automated annotator accepts.

O2. Automatically annotate the documents. The knowledge engineer loads the annotation rules, the catalog of terms, and the batch of text files to the automated annotator. The automated annotator applies the rules and annotate the architectural issues in the text files. The output of this step is a list of sentences that automated annotator suggests as architectural issues. Besides, the knowledge engineer also receives a list of sentences that automated annotator doesn't consider as architectural issues.

O3. Manually fine-tune the results. The knowledge engineer in this stage looks through the list of tool suggestions and reject the sentences that (s)he doesn't consider as architectural entities.

O4. Generate the design guide out of annotated text. Now that the annotated sentences from all of the text files are finalized, guidance generator merges them and produce design guide for a specific project. It includes all of the potential issues (decisions required) in the project. The knowledge engineer can shorten the sentences, classify issues into sub-projects and add alternatives (including pros and cons for each alternative) to each issue. (S)he can also remove the redundant issues.

M1. Suggest new terms for catalog of terms. In step O3, the knowledge engineer may find some terms that would be an indicator for annotating architectural issues. In that case (s)he can suggest them to catalog enhancer.

M2. Add new terms to the catalog of terms: The catalog enhancer conducts the sensitivity test for the suggested term and if the average f-measure is positive, the term will be added to the catalog of terms. So the framework evolves during projects.

3 Framework Evaluation

The main purpose of developing SADGE is to reduce the efforts that manual approach of decision identification technique demands. However, the accuracy and sensitivity of the framework should not be too lower than manual approach; otherwise the framework will not be effective. Therefore these three quality attributes of the framework should be evaluated: processing time (effort), sensitivity and accuracy. The metrics we use for evaluating the effort is time and for evaluating the other two attributes we use the classical metrics in information retrieval domain, recall and precision. In the current stage of the research, we preliminary evaluate the framework by conducting an experiment with students. In the following sections, we first describe the design of the experiment, then we present the results of the experiment and in the discussion section

we interpret and discuss the results and describe the potential threats to validity.

3.1 Evaluation Design (Setup)

Participants: We asked students of a bachelor's program in information technology to participate the experiment. They are familiar with the software architecture. However, they were not familiar with the concept of architectural knowledge (including architectural issues). 19 students (randomly selected) participated in the experiment. We divided them into two groups of ten and nine students. Before the experiment, an introduction about the task and the concept of architectural issue were presented to the students.
Stages: In the first stage, students were supposed to read a text carefully and annotate the sentences they think are architectural issues. In the second stage, the list of automatically annotated sentences from the text given to group 1 in the first stage was given to the students of group 2 and vice versa to avoid the testing effects. They were asked to reject the sentences they disagreed with automated annotator to fine-tune the results.
Material: In the first stage of the experiment each group received two pages of a text from a book chapter on web application design guidance. The texts of two groups are not identical. The book chapter is one of the documents we had used to evaluate the automated annotator (automatic part of the framework). The reason we chose this document among all of the tested documents was that this document had the smallest deviation from the mean of precision and recall of annotating all of the documents and therefore can be considered as a representative of the tested documents.

3.2 Evaluation Results

Table 1 summarizes the results of the experiment. In the manual approach, the students spent nine minutes on average to annotate the architectural issues. The automated annotator ran the annotation procedure in two sec. In the semi-automated approach, the students spent 3 minutes on average to reject those sentences they didn't agree is an architectural issue (we neglect the two second that automated annotator ran the procedure).

Table 1. Results of experiment

Approach	Time (min)	Recall (%)	Precision (%)
Manual	9	38	25
Automatic	**0.03**	**86**	57
Semi-automatic	3	55	**62**

To calculate the recall and precision we needed reference texts. Two experts in the software architecture domain annotated the two texts and the annotated texts were used as the reference text. The recall and precision for all three approaches are presented in Table 1. In the next section, we analyse the results and discuss about their validity.

3.3 Discussion

As Table 1 shows, automated annotator has the highest effort reduction (lowest annotation time) and the highest recall while semi-automatic approach has the highest precision. The effort reduction results are in correspondence to our expectation. Regarding the precision and recall, in the real projects we expect that when those practitioners who are experts in the software architecture domain annotate a text manually, both precision and recall should be near to 100 percent, because the practitioners' knowledge are almost the same as our reference experts' knowledge. Whereas here the results show that the recall and precision of student annotations are very low (38 and 25 respectively). The results for automated annotator are relatively high (86 and 57) and these show that if the people in charge of enhancing architectural guidance are not expert enough, automated approach will perform more accurate and more sensitive by spending much less time. We expected that the semi-automatic approach has the highest precision rate that is in line with the experiment results; but we expected higher precision rate.

The other result that doesn't meet our expectation is the recall rate of semi-automatic approach. Although it cannot be higher than the automatic approach (because some of the positive results have been already neglected by the automated annotator) we expected that the recall would not be lower than the automatic approach. But the results show that some of the positive results are rejected by the participants. This might be caused by the expertise level of the participants. Our expectation is that if the participants were expert enough in the domain, the semi-automatic approach would have almost the same recall rate as the automatic approach and much higher precision rate than the automatic approach. However, this hypothesis needs to be investigated with subject matter experts.

Threats to validity: The potential threat to the internal validity of the evaluation is the testing effect [4]. To avoid the issue, as we explained we divided the participants into two groups and swapped the two documents between the groups. As a result the group 1 in the second stage examined the sentences that group 2 had in the first stage and vice versa.

The potential threat to the external validity of the research is the selection of the material for the experiment because one document cannot be enough for generalizing the evaluation of the framework. We were aware of this issue but to evaluate the framework by applying it on diverse documents we would need to ask students to stay much longer for the experiment that was not feasible. Also as we explained before this document has the smallest deviation from the mean of accuracy and sensitivity of annotating several documents that we tested the automated annotator on.

4 Related Work

Using NLP for knowledge extraction is not novel in software engineering. For instance several researchers have developed tools and methods for generating object oriented models from natural language texts by applying NLP [3][9][10]. However, most of

these methods and tools are applied on specific software documents such as design documents and requirements specifications while more general or informal texts such as meeting minutes, wikis and industrial standards are not considered. Besides, the majority of work has been done to extract the object oriented data from the documents whereas extraction of architectural knowledge (specifically architectural decisions) is not mainly in focus. Even though, there is still few work focusing on architectural knowledge extraction. Figueiredo et al. have developed a rule-based NLP approach to search architectural knowledge entities in documents [7]. TREx is another approach that annotates architectural related documents by applying NLP to retrieve architectural knowledge entities (including issues, drivers, rationale) [8]. Although the development and operation stages of both approaches are very similar to SADGE, the catalog of terms and annotation rules are not presented in the papers nor publicly accessible. Therefore, it is not possible to replicate the approaches and as a result the comparison is not feasible. So the catalog of terms and annotation rules presented in this paper are the contribution of our research to extracting architectural issues from documents and generating architectural guidance.

5 Conclusion and Future Work

In this paper we introduced Semi-Automated Design Guidance Enhancer (SADGE), a framework for obtaining design guidance from architectural knowledge in project documents and domain literature. SADGE applies Natural Language Processing (NLP) to the architectural knowledge domain to reduce the efforts of manually creating architectural guidance from architecture documentation. More specifically, SADGE automatically annotates (highlight) the architectural issues to reduce the knowledge engineering effort that has to be invested initially to identify architectural knowledge from the documents.

We presented the five development stages of SADGE, D1 initializing the annotation rule, D2 initializing the catalog of terms, D3 evolving the annotation rule, D4 evolving the catalog of terms, and D5 refining the catalog of terms. This makes the design of the framework replicable for researchers.

The four operation steps of the SADGE are preparing documents for annotation (O1), automatically annotate the documents (O2), manually fine-tune the results (O3), generate the design guide out of annotated text (O4). The two maintenance steps of the framework are (M1) suggest new terms for catalog of terms and (M2) add new terms to the catalog of terms. This makes the application of the framework understandable for practitioners.

The results of the framework evaluation are: the automatic approach has the shortest processing time (research question RQ1) and the highest sensitivity (RQ2) while the semi-automatic approach has the highest accuracy (RQ3). In summary, using NLP in the architectural knowledge domain reduces the amount of manual decision identification work and has the potential to improve existing decision identification techniques.

Practitioners can use SADGE in the first stages of their architectural decision making

process to rapidly identify architectural issues (decisions required) that are relevant to their project. This helps them accelerate the orientation in the problem-solution space and, consequently, to make architectural decisions in a more confident way.
In the next stage of our research, we plan to improve the sensitivity and accuracy of the automated annotator by applying machine learning algorithms (so far, we were missing adequate training data, but we expect to receive more architectural related documents from real projects in the industry). Furthermore, we plan to evaluate the framework by conducting case studies that involve expert architects and also include more real-world project documents.

References

1. Anvaari, M., Conradi, R., and Jaccheri, L.: Architectural Decision-Making in Enterprises: Preliminary Findings from an Exploratory Study in Norwegian Electricity Industry. European Conference on Software Architecture (ECSA 2013), pp. 162-175, Springer Berlin Heidelberg, (2013).
2. Babar, M. A., Dingsøyr, T., Lago, P. and van Vliet, H.: Software Architecture Knowledge Management, Springer (2009).
3. Bajwa, Imran Sarwar, Ali Samad, and Shahzad Mumtaz.: Object Oriented Software Modeling Using NLP Based Knowledge Extraction." European Journal of Scientific Research 35.01: 22-33 (2009).
4. Campbell D.T. and Stanley J.C.: Experimental and Quasi-experimental Designs for Research. Boston: Houghton Mifflin; (1963).
5. Crowston, K., Liu, X., and Allen, E. E.: Machine Learning and Rule-based Automated Coding of Qualitative Data. Proceedings of the American Society for Information Science and Technology, 47(1), 1-2 (2010).
6. Falessi, D., Cantone, C., Kazman, R., and Kruchten, P.: Decision-Making Techniques for Software Architecture Design: a Comparative Survey, ACM Computing Surveys 43 (4) (2011).
7. Figueiredo, A. M., dos Reis, J. C., and Rodrigues, M. A.: Improving Access to Software Architecture Knowledge: An Ontology-based Search Approach. International Journal Multimedia and Image Processing (IJMIP), 2(1/2) (2012).
8. López, C., Codocedo, V., Astudillo, H., & Cysneiros, L. M. (2012). Bridging the Gap between Software Architecture Rationale Formalisms and Actual Architecture Documents: An Ontology-Driven Approach. Science of Computer Programming, 77(1), 66-80 (2012).
9. Perez-Gonzalez, H. G.: Automatically Generating Object Models from Natural Language Analysis, 17th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications, ACM New York, USA, pp: 86 – 87 (2002).
10. Soeken, M., Wille, R., and Drechsler, R.: Assisted Behavior Driven Development Using Natural Language Processing. In Objects, Models, Components, Patterns (pp. 269-287). Springer Berlin Heidelberg (2012).
11. Tang, A., Avgeriou, P., Jansen, A., Capilla, R. and Babar, M. A.: A Comparative

Study of Architecture Knowledge Management Tools. Journal of Systems and Software. 83, 3, pp.352-370 (2010).
12. Ting, K. M. : Precision and Recall, Encyclopedia of Machine Learning. Springer US, (2010).
13. Tofan, D., Galster, M., and Avgeriou, P.: Difficulty of Architectural Decisions–A Survey with Professional Architects. European Conference on Software Architecture (ECSA 2013), pp. 192-199, Springer Berlin Heidelberg (2013).
14. Zimmermann, O., Koehler, J., Leymann, F., Polley, R., and Schuster, N.: Managing Architectural Decision Models with Dependency Relations, Integrity Constraints, and Production Rules. Journal of Systems and Software, 82(8), 1249-1267 (2009).
15. Zimmermann, O.: An Architectural Decision Modeling Framework for Service-Oriented Architecture Design. PhD Dissertation, University of Stuttgart (2009).
16. Zimmermann, O.: Architectural Decisions as Reusable Design Assets, IEEE Software 28(1), pp. 64-69, (2011).

P4- Rule-based Extraction of Architectural Issues

Submitted for publication

Is not included due to copyright

P5- Associating Architectural Issues with Quality Attributes

Submitted for publication

Is not included due to copyright

APPENDIX B

Supporting Paper

P0- Smart Grid Software Applications as an Ultra-Large-Scale System

Published: In Proc. Innovative Smart Grid Technologies (ISGT), 2012 IEEE PES.

Smart Grid Software Applications as an Ultra-Large-Scale System: Challenges for Evolution

Published: In Proc. Innovative Smart Grid Technologies (ISGT), 2012 IEEE PES.

M. Anvaari, D. S. Cruzes, R. Conradi
Norwegian University of Science and Technology

Abstract. Software applications play a major role to provide the "smartness" in Smart Grid. Such applications are in a never-ending state of flux due to rapidly changing expectations from the users and stakeholders. Hence, managing the evolution is important for development of Smart Grid software applications, and the challenging factors of software evolution in Smart Grid should be identified. This position paper presents a discussion on the Smart Grid software applications as an ultra-large-scale system, focusing on the possible challenges that need to be addressed for an effective and efficient evolution of the area.

Keywords. Smart Grid, software applications, software evolution, systems of systems, ultra-large-scale system.

APPENDIX C

Annotation Rule

The pseudo-code of the annotation rule developed for SADGE was shown in Figure 4.3. The formal version of the rule written in JAPE is presented in this appendix.

```
Phase:  Temp
Input:  Lookup Sentence Split
Options:  control = appelt

Rule: Issue
(
 {Sentence contains {Lookup.majorType == lowissue}}
|
 {Sentence contains {Lookup.majorType == highissue}}
):iss
-->
{
  AnnotationSet annSet = bindings.get("iss");
  Annotation ann = annSet.iterator().next();

  FeatureMap constraints1 = Factory.newFeatureMap();
  constraints1.put("majorType","lowissue");

  FeatureMap constraints2 = Factory.newFeatureMap();
  constraints2.put("majorType","highissue");

  if (inputAS.get(ann.getStartNode().getOffset(),
            ann.getEndNode().getOffset()).get("Lookup",constraints1).size() > 1){

   FeatureMap features1 = Factory.newFeatureMap();
   features1.put("rule", "Issue");
   outputAS.add(annSet.firstNode(),annSet.lastNode(),"Issue",features1);

  } else if (inputAS.get(ann.getStartNode().getOffset(),
            ann.getEndNode().getOffset()).get("Lookup",constraints2).size() > 0){

   FeatureMap features2 = Factory.newFeatureMap();
   features2.put("rule", "Issue");
   outputAS.add(annSet.firstNode(),annSet.lastNode(),"Issue",features2);

  }

}
```

Figure C.1: SADGE annotation rule written in JAPE for use in GATE

APPENDIX D

Experiment Material

The text we used for both groups in the experiment with expert architects are presented in this appendix.

Group 1
Start time (hh-mm): End time (hh-mm):

Text 1 (from a company documentation):

From a product perspective, no differentiation is done between the three packet data networks (2G, 3G and LTE). We only talk about packet data access and give access to all networks - or none. There is however, one exception to this rule. A LTE capable USIM card is required to be able to connect to the LTE network. These cards will be distributed from Q3 or Q4 2011. S212 will only provision LTE access for LTE capable SIMs.

Data volumes are currently increasing steadily and the introduction of LTE will further accelerate the growth. The number of data CDRs created indicates growth of 500% over the next 5 years.

Our target is that roaming agreement and settlement shall be technology neutral. Assumption is therefore that no changes in TAP formats will be required to handle LTE roaming. CDRs coming from the LTE network shall be handled as 'normal' packet-data CDRs. This means that the mediation systems shall produce the same packed data CDRs independent of source network.

IP address fields in CDRs shall support both IPv4 and IPv6 addresses. TAP standards for IPv6 have not yet been defined. When they get defined, LTE and IPv6 project must agree on who should be responsible for implementing the required changes.

There are no big architectural changes in the OSS architecture, but quite a lot of 'LTE upgrades'. The upgrades are mainly triggered by new elements and SW upgrades in the mobile network.

MXX should be master when turing on LTE cells. This is the same as 2G/3G today after swap. The reason is that there will be tuning of the cells for KPI optimization, and this will probably be done in MXX by AMN, thus MI must accept these changes. This subject must be discussed with AXX Corp. and IS architects to view the details and the dates of changes of mastering.

Potential issue related to HLR/HSS migration:

Current assumption is to use IMT series to define the 'migration bulks'. How to handle call set-up during migration (i.e. how to determine in which HLR a subscriber with a given MSIC resides) might be an issue (still no requirements to IS). One resolution to this might be to start provisioning FNR with IIB-DIC mapping. This is not a current requirement. In case IS have to start provisioning IMTs to FNR this will have significant impact on Instant link. This scenario will be analyzed if and when a change request is raised by the migration team. LTE will also migrate the remaining core nodes from OSS-RC 2G and 3G, allowing these two 'old' OSS nodes to be phased out.

Performance management:

Brain OSS will introduce basic Performance Management functionality and support for

RAN and PS core in 22H2. LTE project will make necessary adjustments required when element managers SW are upgraded after Brain OSS is closed down.

Text 2 (from a standard):

For those requiring interoperability with BACnet, this OpenADR specification may optionally use the BACnet Web Services (BWS) specification (ANSI/ASHRAE Addendum C to Standard 135 - 2004) to communicate with BACnet - based systems. The generic BWS data model allows interoperability with DRAS - issued DR event information and to schedule response strategies using Smart and Simple DRAS clients. The DRAS - BACnet Server and supporting DRAS Client should exchange EventState information using two different modes of interaction–PUSH and PULL–as described earlier in "Section 6.5.3.1, Modes of Interaction (PUSH versus PULL)."

While BWS has many services that support various aspects of control systems specifications, most of them are not relevant to the DRAS - BACnet server and getValue and setValue services are being used. The other two services getDefaultLocale and getSupportedLocals are required as per BWS specifications and must also be supported by the DRAS to exchange DR event data with the DRAS - BACnet server.

In general there are many modes of attacks upon any sort of IT infrastructure ranging from intruders gaining physical access to the servers to remotely accessing the servers through open communication channels. This OpenADR specification only covers the communication protocols used to interact with the DRAS and the DRAS Clients. It is therefore only intended to cover modes of attack that would be perpetrated by using one of the communications channels that are used to implement the interface to the DRAS as described in the analysis section of Appendix C. Any other certainly necessary security measures (firewalls, intrusion detection, etc.) are not covered.

There are a number of types of users that require access to the DRAS. Each user may have different requirements on the type of functions they can perform and data they may access. To support limiting the access of the DRAS users based on their requirements, the DRAS must support the security roles outlined in Section 6.3.1. These security roles are designed to limit access to the various methods in each of the Web service interfaces. Table 5 describes how each of the security roles is limited within each of the Interfaces.

An implementation chooses the security measures for the non - API interfaces according to the usage scenario, threat levels, protected values, etc. The minimum level, given in this document, might (Client A) or might not (Client B and C) be right for a particular implementation as examples shown in Figure 45. Higher security measures can easily be integrated into the DRAS if necessary (Client C) as long as they are based on open standards. Communication partners with lower security levels (Client B) have to use a security proxy.

Group 2
Start time (hh-mm): End time (hh-mm):

Text 1 (from a company document):

There is no 'prioritization' functionality in the provisioning gateway. This means that any prioritization strategy must be implemented in the provisioning system.

Both IL and EMA have the option to define the maximum allowed number of parallel connections to PG. This feature can be used to prevent one of the systems (most likely IL) to consume all the capacity at PG. Combined with setting priority on different orders in IL and EMA it should be possible to ensure that critical orders are processed quickly also at peak traffic.

Only data is in scope for LTE initially. CDRs coming from the LTE network shall be handled as 'normal' packet-data CDRs. This means that the mediation systems shall produce the same packed data CDRs independent of source network (2G, 3G or LTE).

It must be discussed how the top-up components can identify both the top-up service and the corresponding CAN/FQP-service in a way which is available for NIRRC. One very simple possibility is to use the prefixed system-component-id of the CAN/FQP component as the system-component-id of the top-up component.

The additional quota value will be available as a configuration which is set when defining the product. The component will have configurations which indicate the end-date to be used when activated is the last day of the current month.

There have to different solutions for handling this optimization for products in the old (Subscriptions) and new (Add-on products) Product Catalog. The optimization for the old PK is basically handled in S212 while the optimization for the new PK spans several systems.

Maximum uplink and downlink speed must be present in all CDRs (Billing Gateway). SGTTN and GGTRSN addresses are used for charging purposes today. The introduction of SGW and PGW node will impact this regime. Correct mapping of SGW and PGW addresses must be implemented by Billing Gateway or EPP.

The implementation of new MBB products for SP will increase the number of orders in the mobile value chain. Estimate from wholesale is that approx 15% of the SP customers will have a MBB product. This should result in an increase of 5000 orders per month.

The optimization of PCRF orders will decrease the number of tasks towards PCRF with 65 000 per month.

Migration of all subscribers from current (classic) HLR to new UDR is described in the LTE Technical AO. The migration will be done in bulks of approximately 3 mill subscribers – indicating that 4-8 bulks will be required to migrate all our subscribers.

The LTE component (or EPS as the service is named in HSS/CUDB) will be added after migration of all subscribers. This is done to minimze the risk of the 2G/3G migration. Instant link will be the only system in IS Mobile value chain that 'see' the migration.

Text 2 (from a standard):

The DRAS must support two-way communications for both the PUSH and the PULL model of interaction, but the DRAS Client is only required to support one or the other. Typically the PULL model may be used since the DRAS Client has more control over the communications including the ability to more easily communicate through firewalls and being network-friendly.

When a program is defined within the DRAS there are specifications associated with the program that define what type of information may be associated with a DR event when one is issued for that program. Each type specification for an EventInfoInstance is referred to as an EventInfoType. A program may be defined that allows for multiple different types to be associated with a program.

The Operation Mode variable takes on values according to a schedule during the event that is defined by the participant or the utility or ISO. This schedule is specified by using a set of rules that determine how the EventInfoInstance of the UtilityDREvent is translated into one of the simple values of the operation mode. Since the participant is free to schedule how the Operation Mode variable changes, this defines a so called "Response Schedule" for how that participant responds to DR events. The response schedule is represented by the ResponseSchedule entity.

The DRAS is responsible for tracking the event states for each of the DRAS Clients in order to send the DR event information to the DRAS Client at the appropriate time.

From the DRAS Client's point of view there is a so-called DR event state the DRAS Clients are in which is represented by the EventState entity. Normally a DRAS Client's event state is "IDLE" meaning that there are currently no active or pending DR events. This changes when the utility or ISO initiates a DR event in the DRAS. The DRAS tracks the DR event state for each DRAS Client and can provide the current state information at any time for that DRAS Client. It can be in different states, depending upon whether the participant uses a Smart DRAS Client or a Simple DRAS Client.

This subsystem is responsible for notifications to facility operators using various existing technologies such as phone, pages, email, fax. The purpose of showing this as a separate component is to highlight the fact that certain types of notifications (such as voice mail) will not be part of the specification and may be provided by third party systems. The systems may be part of the utility infrastructure, but in the most general case they are a standalone service as depicted in the diagrams. At a minimum, the DRAS must support direct email notification to the facility that includes exception handling and bidding information.