university of groningen

faculty of mathematics and natural sciences

Tool-support for Architecture Decision Documentation

Christian Manteuffel
Spyros Ioakeimidis

Groningen, August 30, 2013

University of Groningen
Faculty of Mathematics and Natural Sciences
Department of Mathematics and Computing
Science Software Engineering and Distributed Systems

Master's Thesis
August 2013

Student: Christian Manteuffel
Student: Spyros Ioakeimidis

Supervisor: Prof. dr. ir. Paris Avgeriou
Second Supervisor: Prof. dr. ir. Marco Aiello

Abstract

The perspective of looking at software architecture as a set of architecture decisions is widely recognized in research. However, in practice architecture decisions are often not explicitly documented but reside in the architect's mind as tacit knowledge or are only implicit in the models that the architect creates. Even though explicit capturing and documentation of architecture decisions has been associated with a multitude of benefits such as avoiding knowledge vaporization, supporting change impact estimation, increasing system understanding and knowledge sharing, and facilitating architecture evaluation.

In cooperation with a large multinational industrial corporation, we developed a tool that should facilitate the documentation of architecture decisions, as there is currently no standardized decision documentation approach or tool that is suitable for company-wide use. The tool is an extension to Sparx Systems' Enterprise Architect and is based on a theoretical framework comprised of five architectural viewpoints. The framework has been developed by the Software Engineering and Architecture group of the University of Groningen.

We conducted an exploratory case study at our industry partner. The objective of this study was to assess the status quo of architecture decision documentation, to identify expectations of the ideal documentation approach, and to evaluate the advantages and limitations of the developed extension.

The findings of the case-study suggest that the developed extension allows documenting decisions in a user-friendly way and provides valuable support to capture architecture decisions. However, we also identified important improvements, such as a clear separation of problem, outcome and alternatives within the used meta-model. The study also showed that decisions are receiving rising awareness but that a lack of tool-support and a lack of standard approach for ADD limit the documentation and the use of decisions in practice.

Acknowledgements

First of all, we would like to express our special thanks to our supervisor Paris Avgeriou, for his professional guidance and valuable support, and to Dan Tofan for his advice and constructive feedback. Special thanks should be given to Heiko Koziolek for his effort to make this collaboration possible and his constructive suggestions on how to improve the tool.

We wish to acknowledge the help provided by Werner Buck and Zengyang Li by taking part in the pilot study and for giving us important feedback to improve the study design. We would also like to thank the participants of our case study.

Last but not least, we wish to thank our families for their support and encouragement throughout the master's project.

Contents

Abstract II

Acknowledgements III

Contents IV

List of Figures VI

List of Tables VII

Glossary VIII

1. Introduction 1
- 1.1. Background . . . 1
  - 1.1.1. Software Architecture . . . 2
  - 1.1.2. Descriptions and Viewpoints . . . 3
  - 1.1.3. Architecture Decisions . . . 4
  - 1.1.4. Decision Documentation Framework . . . 5
- 1.2. Related Work . . . 8
- 1.3. Problem statement . . . 10
- 1.4. Approach . . . 11

2. Analysis, Design and Implementation 12
- 2.1. Domain . . . 12
- 2.2. Use-cases . . . 13
- 2.3. Requirements . . . 14
- 2.4. Key Drivers . . . 14
- 2.5. Architecture of the Add-in . . . 15
  - 2.5.1. Logical View . . . 16
  - 2.5.2. Implementation View . . . 18
  - 2.5.3. Deployment View . . . 22
  - 2.5.4. Design Decisions . . . 23

3. The Add-in 31
- 3.1. Integration into Enterprise Architect . . . 31
- 3.2. Decision Views . . . 34
  - 3.2.1. Detail . . . 34
  - 3.2.2. Relationship . . . 35
  - 3.2.3. Chronology . . . 36
  - 3.2.4. Forces . . . 36
  - 3.2.5. Stakeholder Involvement . . . 38
- 3.3. Additional Features . . . 40
  - 3.3.1. Tracing . . . 40

3.3.2. Model Validation . . . 41
3.3.3. Reporting . . . 41

4. Evaluation **44**
4.1. Research Questions . . . 46
4.2. Case and Units of Analysis . . . 46
4.3. Design . . . 47
4.4. Data Collection . . . 48
4.4.1. Requirement Elicitation Interview . . . 48
4.4.2. Existing decision documentation . . . 49
4.4.3. On-site Study . . . 49
4.4.4. Subjects and Projects . . . 50
4.5. Analysis . . . 51
4.5.1. Analysis Procedure . . . 51
4.5.2. Categories . . . 52
4.5.3. RQ1 - Status-Quo of Decision Documentation . . . 53
4.5.4. RQ2 - Ideal AD Documentation Approach . . . 55
4.5.5. RQ3 - Advantages and Limitations of the Add-in . . . 56
4.6. Limitations . . . 57
4.6.1. Construct Validity . . . 58
4.6.2. External Validity . . . 58
4.6.3. Internal Validity . . . 59
4.6.4. Reliability . . . 59

5. Discussion **60**
5.1. Lack of Tool-Support and Standard Approach . . . 60
5.2. Sharing and Reusing AK . . . 60
5.3. Efficiency, Flexibility and Perceived Benefits . . . 61
5.4. Reinforced Findings about the Framework . . . 62
5.5. Separation of Problem, Alternatives and Outcome . . . 62
5.6. Features and improvements of the add-in . . . 63

6. Conclusion **65**

References **67**

Appendices **72**

A. Life-cycle of a decision **72**

B. Interview Guidelines **73**
B.1. Requirements Elicitation Interview Guide . . . 73
B.2. Pre Tool Session Interview Guide . . . 74
B.3. Post Tool Session Interview Guide . . . 75

C. Software Requirements Specification 77

D. Example Decision Documentation 83

E. Backlog 89

List of Figures

1. Decisions viewpoints and their relationships. 6
2. Meta model of the decision documentation framework. 8
3. Project phases and milestones. 11
4. The component view of the add-in. 16
5. The high level implementation view of the add-in. 18
6. DecisionViewpoints library. 19
7. Handlers within the DecisionViewpoints library. 20
8. EAFacade library. 21
9. DecisionViewpointsCustomViews library. 21
10. Add-in architecture of Enterprise Architect. 22
11. The deployment view of the add-in. 23
12. Storing decision state changes. 24
13. Creating a Forces view. 25
14. Reporting mechanism. 26
15. Accessing the Enterprise Architect API. 28
16. Storing the rationale contained in Detail view. 29
17. Generating the Chronology view. 30
18. The user interface of Enterprise Architect. 31
19. The context menu of the add-in. 32
20. The project browser showing elements of the add-in 32
21. Viewpoint-specific toolboxes. 33
22. Detail view in Enterprise Architect. 35
23. Relationship view in Enterprise Architect 36
24. Chronology view in Enterprise Architect. 37
25. Configuration perspective on the Forces view. 37
26. The table part of the Forces view. 38
27. Stakeholders involvement before Minor Milestone 2. 39
28. Stakeholders involvement after Major Milestone 1. 39
29. Trace between decision and baseline component. 40
30. Following a trace through the context menu. 41
31. Generated Word document. 42
32. Generated Excel document. 42
33. Generated PowerPoint document. 43
34. Case Study process. 45
35. Schema of an individual session. 49

36. A meta-model that supports problem, alternatives, and outcome. 63
37. Life-cycle of a decision architecture decisions. 72

List of Tables

1. Mapping between sources of data and RQs. 47
2. Mapping of categories to sources of evidence. 52

Glossary

Add-in A software application that extends the capabilities of existing software systems.

Architecting Process Comprises of various architecture activities performed to synthesize, analyze, and evaluate a software architecture.

Architecture Description Practices for expressing, communicating and analyzing software architectures.

Architecture Review The process of determining how well the current design or a portion of it satisfies the requirements derived during analysis.

Case-study A descriptive, exploratory or explanatory analysis of a person, group or event.

Case-study Protocol A predefined written procedural method of conducting case studies.

Force Anything that has a potential nontrivial impact of any kind on an architect when making decisions [1].

Knowledge Management A range of strategies and practices used in an organization to identify, create, represent, distribute, and enable adoption of knowledge.

Meta-model A simplified or approximated descriptive model of another descriptive model.

Software System A system of intercommunicating components based on software forming part of a computer system.

Stakeholder An individual, team, organization that have an interest in a system or are affected by it [2].

View Document or model expressing the architecture of a system from the perspective of specific system concerns [2].

Viewpoint Establishes the conventions for the construction, interpretation and use of views [2].

Acronyms

AD Architecture Decision.

ADD Architecture Decision Documentation.

AK Architectural Knowledge.

API Application Programming Interface.

BU Business Unit.

EA Enterprise Architect.

IEC International Electrotechnical Commission.

IEEE Institute of Electrical and Electronics Engineers.

ISO International Organization for Standardization.

RGT Repertory Grid Tool.

SRS Software Requirements Specification.

UML Unified Modeling Language.

1. Introduction

The perspective of looking at software architecture as a set of architecture decisions is widely recognized [3, 4, 5]. However, in practice architecture decisions are often not explicitly documented but reside in the architect's mind as tacit knowledge or are only implicit in the models that the architect creates [4, 6]. Even though explicit capturing and documentation of architecture decisions has been associated with a multitude of benefits such as avoiding knowledge vaporization, supporting change impact estimation, increase system understanding and knowledge sharing, and facilitate architecture evaluation [4, 6, 7].

This gap between industry practice and the benefits demonstrated by research persists, notwithstanding the fact that research has produced a significant number of approaches to find a suitable representation of architecture decisions and to "incorporate the documentation of architecture decisions in architecture practice [8]". However, in previous research, van Heesch et al. argued that the proposed approaches for documenting architecture decisions, i.e. decision templates, decision models, and annotations, are not sufficient as they do not address holistically the concerns of all stakeholders. They stated that "there is no single way of documenting architecture decisions that frames all concerns of all stakeholders in an adequate and useful manner". Therefore, they proposed a decision documentation framework which consists of multiple dedicated viewpoints to focus on specific decision-related concerns [5, 9].

This framework is designed to support the complete architecting process. However, currently there is no tool-support adopted by the industry.

This thesis is organized as follows. Section 1 discusses the theoretical foundation on which the tool is based and presents related work. Section 2 describes the activities undertaken during the problem analysis and discusses the tool's architecture. Section 3 illustrates the use of the tool in practice. Section 4 reports on an industrial case study, which was conducted to evaluate the tool. Section 5 presents the findings of the evaluation. The thesis ends with conclusions and directions for future work in Section 6 .

1.1. Background

This section illustrates the background of the project. First, the concept and challenges of Software Architecture are explained. Afterwards, documentation of Software Architecture and the ISO/IEC/IEEE 42010 standard are briefly illustrated. Subsequently, the section discusses Architecture Decisions, their benefits for Software Architecture but also current challenges in the field. Afterwards, the Decision Documentation Framework is presented.

1.1.1. Software Architecture

Software architecture can be described by using an analogy from construction. Buildings serve a functional purpose but also need to adhere to cross-cutting properties such as the overall energy consumption, climate system, evacuation plans, or aesthetics. Furthermore a building needs to be realized within financial boundaries. These fundamental properties of a building are determined by its architectural design. Just like buildings, a software system has functions, qualities, and budget. For example, a typical function of a browser is serving webpages and a quality would be how fast it can display webpages or how secure it is. Both functions and qualities are influenced by the architecture of the software, hence the architecture determines how well a system is able to fulfill its needs. However, unlike a building's architecture the architecture of a software system is not directly visible and therefore cannot be observed by looking at the final application. This makes software architecture an abstract concept and as a result of this there is a multitude of definitions for software architecture. One definition in the ISO/IEC/IEEE 42010 describes 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 .

This definition emphasizes that the architecture is not necessarily everything about a system but only the aspects that are fundamental to a system[1]. This definition does not take a pure product-oriented perspective on software architecture. "Apart from elements, relationships, and properties, it takes the principles into account that govern the design and evolution of a system. A principle is not necessarily of technical nature. An example for a non-technical principle could be: maximize financial benefit to the company [9]."

With the growing size and increasing complexity of software systems, designing these systems has become increasingly difficult. However, just like any other complex system, software systems must be built on a solid foundation. The risks caused by poor architecture design are ranging from unstable systems and high maintenance costs to failed projects and completely unusable systems.

Software architecture as an activity provides means to manage the complexity of a system, which includes among others balancing multiple possibly conflicting goals and concerns; involvement of and negotiation among multiple stakeholders [10]. Several architectural design methods has been defined and are used in practice, such as Attribute-Driven Design or Siemens' 4 view model. According to Hofmeister et al. [10], the different design methods can be reduced to three core architecting activities: analysis, synthesis, and evaluation. These activities are not performed sequentially, but rather iteratively as "architects move constantly from one to another, growing the architecture progressively over time" [10]. Architectural analysis can be described as problem-space investigation. The

[1] http://www.iso-architecture.org/ieee-1471/defining-architecture.html [Online accessed: 28/08/13]

architect needs to identify the problems that the architecture must solve and identify the concerns and requirements that are architecturally significant. Architectural synthesis describes the process of identifying solutions to the problems that were elicited during analysis and combining these solutions into a candidate architecture. The candidate architecture is assessed during architectural evaluation, which ensures that the decisions made by the architect are the right ones.

It can be seen that decisions play an important role in software architecture as architecting can be seen as a decision-making process. In the last decade the perspective to see software architecture as a set of architectural design decisions emerged [3, 4, 5]. The previous perspective on software architecture focussed primarily on the final product, omitting the rationale that has led to the architecture [11] resulting in all sorts of problems, such as knowledge vaporization or increased maintenance costs.

1.1.2. Descriptions and Viewpoints

Every software system has an architecture but not necessarily a description or specification of it [12]. Architecture descriptions are used to communicate and share a conceptualization of a system's architecture with stakeholders. As pointed out by Bass et al. [12], a software architecture is a complex entity that cannot be described in a simple one-dimensional fashion. Therefore, the concept of viewpoints and views acknowledges the idea that there are different ways of looking at a system and allows to separate concerns by creating stakeholder-specific views. A viewpoint is a specialized perspective onto the architecture that frames one or more stakeholder concerns. However, the viewpoint is only an abstract convention for constructing, interpreting, and analyzing concrete views. A view is a concrete instance of a viewpoint and is governed by the viewpoint's conventions.

The ISO/IEC/IEEE 42010 standardizes best practices for software architecture descriptions, which is the successor of the IEEE Std 1471. The standard provides an ontology for the description of architectures and serve to enforce desired properties of architecture descriptions. The standard formalizes the concept of viewpoints, frameworks and description languages. The 42010-standard embodies the concept of architectural knowledge via rationale and decisions. Architectural rationale "records the explanation, justification or reasoning about Architecture Decisions that have been made and architectural alternatives not chosen [2]".

1.1.3. Architecture Decisions

The paradigm shift towards architecture decisions as first-class entities in architecture descriptions, rather than just interacting components and connectors, is mainly motivated to avoid knowledge vaporization [7]. Knowledge evaporates when it is not explicitly documented but is only implicitly kept in the mind of the architect. Hence, knowledge stays tacit and in case the architect leaves the project or simply forgets, the reasons that have led to the system's design are lost. This is in particular a problem during the evolution of a system. However, as the definition of software architecture is ambiguous as there is no consensus of what an architectural significant decision is. As described by Tyree and Akermann an indicator is ,if a decision affects one or more system qualities [6]. Kruchten proposes a classification and thereby narrows down the concept of architectural significance. According to this taxonomy, decisions are categorized into existence decisions, about the inclusion or exclusion of a structural or behavioral element into the system; property decisions about design rules or constraints; and executive decisions that are driven by business or affect the development process or people [11].

Architecture decisions documentation is the process of making the rationale explicit, which lead to a decision. While there is no limit to the information that can be documented, a small set of attributes has been identified as being the standard elements of an architecture decision, this includes the decision outcome, alternatives (solution), the rationale and constraints [13].

The knowledge embodied in explicitly documented decisions can support the architecting process beyond knowledge vaporization. For instance, weaknesses in the chain of reasoning can be identified more easily as the rationale is available. Furthermore, it can lead to more mature design decisions as it is easier to evaluate if all constraints and possible solutions are considered. In the same way decisions provide a solid basis for architecture reviews and evaluations. The more complete the documentation is, the stronger the possible benefits for the architecting process.

Architecture decisions should be documented to complement architectural design with rationale. There are currently various approaches for architecture decision documentation that try to fulfill this requirement. These approaches have been briefly presented in [5].

Decision templates: A variety of templates have been proposed to document architecture decisions using textual form. One of them is the template by Tyree and Akerman [6]. These templates can be used to capture rationale behind decisions, providing also information about related decisions and requirements, assumptions, constraints, and the status of the decision. Their tabular form allows for an easy integration into third party tools, such as word processors and sheets. They also provide flexibility to documenters as to what kind of information to document and what to leave out.

Annotations: Other approaches use annotations to document architecture decisions, for instance the Knowledge Architect Suite by Liang et al. [14]. Users can assign comments to other Unified Modeling Language (UML) or Architecture Description Language elements, or to specific pieces of text. They highlight elements as decisions and additionally capture relations, attributes, and the history of decisions. An advantage of these approaches is that they are not tied to specific tools and different stakeholders can use them with different tools to attach annotations. Moreover, annotations provide an well-suited way to elicit decisions from existing documentation.

Decision models: These models can represent the same information as templates and annotations. However, they use dedicated models to represent decisions and address decision-related concerns specific to particular views [4, 15, 16]. For example a decision model in a component view, would capture decisions about the allocation of components on specific system parts. This approach provides a better overview of decisions than the aforementioned approaches of template and annotations and facilitates linking decisions to other architecture description elements.

According to van Heesch et al., the aforementioned approaches for architecture decision documentation exhibit certain limitations [5]. For example, it is very easy to lose the overview of decisions when many of them are documented for one system. Furthermore, with most of these approaches it is challenging to identify relationships between decisions, and to perform change impact analysis. They do not provide direct support for lifecycle management and consistency checks. Moreover, they are dedicated to existing architectural viewpoints, addressing system concerns, and they cannot address additional concerns specific to architecture decisions. The documentation framework for architecture decisions was designed to overcome most of these limitations.

1.1.4. Decision Documentation Framework

The documentation framework for architecture decisions [5] and its extension [1] enable the creation of intuitive decision models that can capture implicit architectural knowledge. It uses the conventions of the international standard for architecture description ISO/IEC/IEEE 42010 [2]. The framework's compliance with ISO/IEC/IEEE 42010 allows to combine it with other viewpoint based architecture descriptions such as Kruchten's 4+1.

The decision framework consists of five viewpoints, each being dedicated to satisfying different stakeholder concerns related to architecture decisions. As illustrated in Figure 1 the viewpoints are not independent but share information with each other. We briefly describe each viewpoint including related stakeholders and concerns.

Detail Viewpoint

The decision detail viewpoint combines the information shown in all other views by giving detailed information about single architecture decisions including their rationale. It uses a set of common attributes, such as the state of the decision, issue, alternatives, and related concerns. It also aggregates all incoming and outgoing relationships of a decision shown in the relationship viewpoint. Typical stakeholders for this viewpoint are reviewers, architects, customers, managers, new project members, and requirements engineers. The detail viewpoint addresses a large set of concerns, like conflicting impacts between decisions, dependencies, relations, the involvement of stakeholders, and the history of a specific decision.

Relationship Viewpoint

The decision relationship viewpoint makes relationships (e.g. causality, dependency) between architecture decisions explicit. It shows decisions, their relationships to other decisions, and their current states. It addresses concerns mostly related to change impact and dependency analysis. It is well-suited to get an overview of current set of decisions. Stakeholders related to concerns of this viewpoint are architects, reviewers, and domain experts.

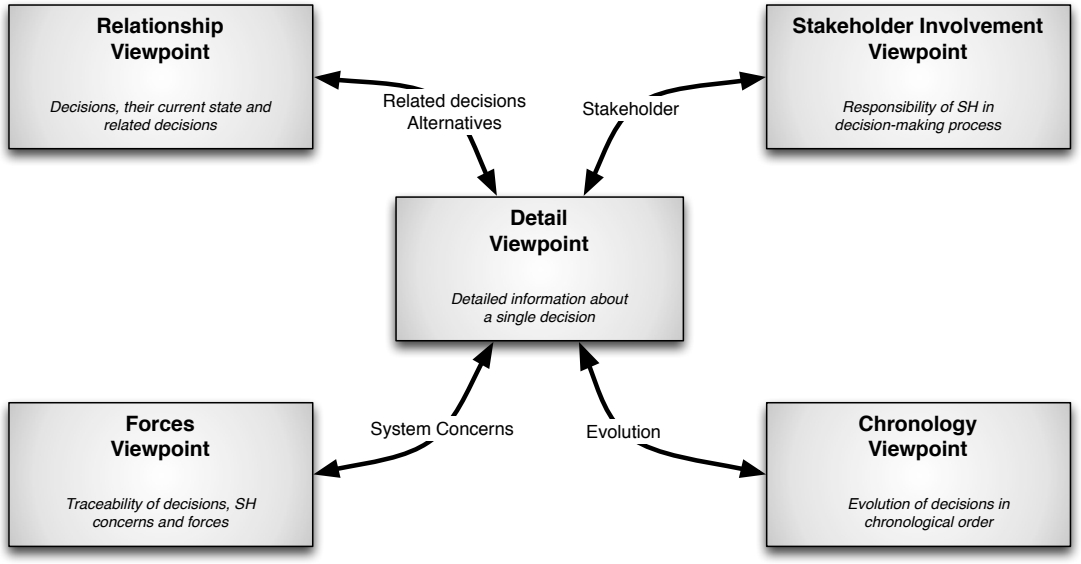


Figure 1: Decisions viewpoints and their relationships.

Chronology Viewpoint

The decision chronology viewpoint shows the evolution of architecture decisions over time. It shows decisions that have been made, current set of decisions, the ordering of decisions, and obsolete decisions. Typical stakeholders for this viewpoint are reviewers, architects, and new project members.

Stakeholder Involvement Viewpoint

The stakeholder involvement viewpoint explains the responsibilities of specific stakeholders in the decision-making process. It depicts decisions, actions, and stakeholders involved within one architecture iteration. In many cases knowledge becomes tacit and the people that hold the rationale behind a decision are either leaving the project, or worse quit the company. The role of this viewpoint is not to show the rationale behind a decision but the involvement of people in the decision-making process. Typical stakeholders for this viewpoint are reviewers, architects, and managers.

Forces Viewpoint

The decision-forces viewpoint makes relationships between decisions and decision forces explicit. The decision-related concerns framed by this viewpoint are identifying the impact of forces on decisions, conflicting influence of forces on a specific decision, rationale behind a decision, and to what concerns a decision pertains. Typical stakeholders for this viewpoint are architects, reviewers, requirements engineers, new project members, and domain experts.

At the same time, each viewpoint is integrated with the other viewpoints through a common meta-model to offer a more complete picture of decisions and their rationale. The common meta-model for the decision elements is illustrated in Figure 2. The central element in the model is the Architecture Decision. A decisions has a life-cycle indicated by a state. The following decision states are defined in the framework: idea, tentative, discarded, decided, approved, challenged, and rejected. A state machine is implied by the framework, which can be found in Appendix A. The model allows to create relationships between decisions using one of the following relationship types: depends on, caused by, is excluded by, replaces, and is alternative for.

Additionally, inter-model and inter-view correspondence rules are defined to ensure consistency between the views. For example, the history of a decision represented in the decision detail model must contain all stakeholder actions performed on that decision shown in all stakeholder involvement models. Another example is that the alternatives mentioned in one decision detail model must be identical to the decisions in the relationship view having an is alternative for relationship pointing to the decision represented in the model.

Parts of the decision documentation framework were implemented in the Open Decision Repository project[2]. The goal of this project was the development of a web application that provides the functionality to document and visualize architectural design decisions for software-intensive systems.

[2] https://code.google.com/p/opendecisionrepository/ [Online accessed: 6/9/2013]

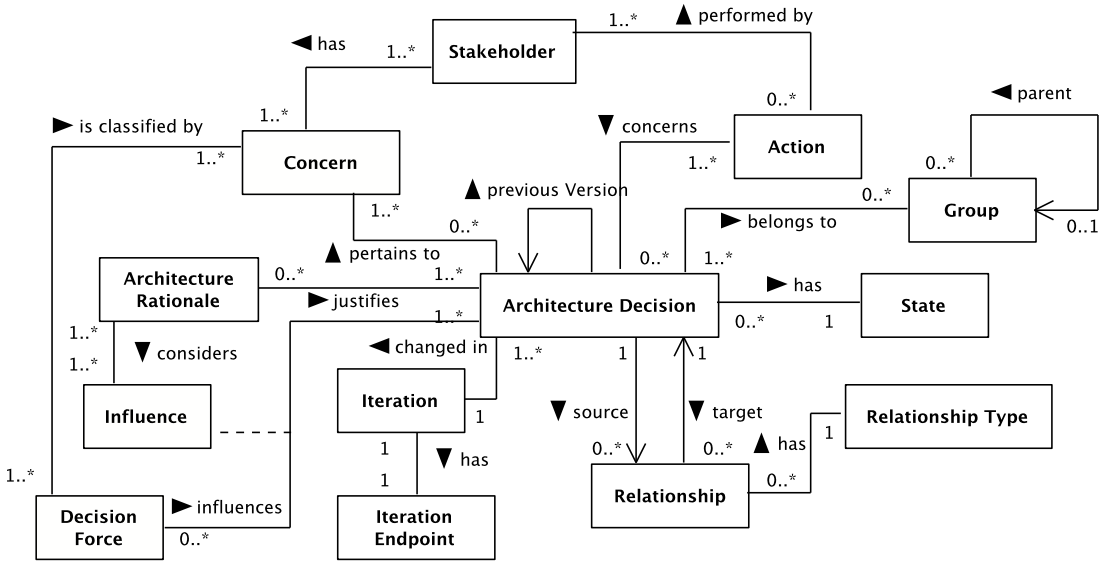


Figure 2: Meta model of the decision documentation framework. Reproduced from [9].

1.2. Related Work

In the past ten years, different approaches emerged to support architectural knowledge management activities that were using design decisions as basic knowledge entities. However, each approach proposes its own notations and emphasize on different architectural life-cycle phases and knowledge management activities. Several studies were performed to assess the advantages and limitations of the various approaches and tools from different perspectives.

Tang et al. conducted a study that compared five architectural knowledge management tools based on a comparison framework that comprised ten distinct criteria that represent how Architectural Knowledge (AK) management is used in the architecture life-cycle [17]. They found that each tool focusses on certain stages of the architecture life-cycle but there is no comprehensive tool that supports all the activities, i.e. analysis, synthesis, evaluation, implementation, and maintenance. The results also show that the capability to provide different perspectives on AK is limited as the tools generally do not support the concept of views and viewpoints. Furthermore, it was noted that knowledge sharing features are only mentioned as a future work and not fully supported by the tools yet.

This finding is also supported by an earlier study by Farenhorst et al. in 2007 [18]. In this study, the authors analyzed five tools from academia and practice from a knowledge sharing perspective. Therefore, desired properties of AK sharing tools were elicited. The results show that AK personalization strategies are not sufficiently supported and that the tools do not provide the possibility to generate stakeholder-specific content. The latter was also identified by [17].

In 2009, Liang and Avgeriou surveyed existing tool-support for knowledge management based on the offered use cases [14]. The use cases were grouped into four categories: consuming AK, knowledge management, intelligent support, and producing AK. The results listed use cases which are not fully being supported by existing AK management tools, amongst others the ability to identify stakeholders, the application of AK decisions to produce the design (synthesis), the evaluation of AK, and support to assess design maturity. In total nine technologies were evaluated. A study by Shahin et al. analyzed architecture decision models in order to clarify similarities and dissimilarities between existing approaches and tools [13]. They found that all of the nine models that were assessed have a consensus on capturing and documenting rationale, constraints, and alternatives of an architecture decision but use different terms to express identical concepts.

In the following, five tools or tool-suits are presented as an example for the different approaches towards architecture decision documentation. The tools have fundamental differences in the set of functionalities and how they are implemented, e.g. a web application, an extension to a programming language, or an add-in for Word.

The Knowledge Architect is a tool suite, which is based on a common knowledge repository that is accessed by different clients to store and retrieve AK [19]. The tool portfolio includes among others an explorer to analyze and visualize relationships between AK entities; a Word plug-in that supports capturing AK by annotating relevant text passages and using AK from the repository in the document; and an Excel plug-in that supports capturing AK from quantitative analysis models.

The Architecture Design Decision Support System (ADDSS) is a web-based tool storing, managing, and documenting architectural decisions [20]. It does not only capture decisions but also allows to capture candidate architectures. ADDSS focusses on visualizing the evolution of AK over time by capturing AK and architecture in an iterative process. It advocates the use of mandatory and optional attributes for characterizing the design decisions and provides a combined codification-personalization strategy [17]. Furthermore, the tool provides traceability between requirements and design decisions and offers a query system to retrieve information about related requirements, decisions, and architecture artifacts stored in the tool. ADDSS also allows to generate PDF reports containing a detailed description of the architecture and the decisions [14].

Lee and Kruchten propose a three-pronged approach to decision capturing, which are formal elicitation, lightweight top-down capture, and lightweight bottom-up capture [21]. They provide tool-support for each of mentioned approaches targeted for highly-technical architectural discussion sessions, where decisions are created or modified in batches in response to a technical discussion. The formal elicitation tool presents a decision-attribute form to gather software decisions in an explicit and structured manner. The second tool attempts to support software architects and designers in the early design phase, by using digital sticky-notes as bookmarks for later information retrieval. The bottom-up capture tool is implemented as Eclipse plug-in that identifies special decision tags within source code.

Archium is a component language extending the Java programming language[3], which allows to model design decisions and relate those to the resulting components [3]. The language allows an architect to describe the elements from a component & connector view; to express architectural decisions, design fragments, and rationale; and to model the relationship between architectural decisions and the architectural entities [22]. This makes a single language act as an architectural knowledge repository and implementation of a system. Archium consists of a compiler, a run-time platform, and a visualization tool that indicates consequences of a decision. The tool has been applied in several small case studies to demonstrate its application.

PAKME (Process-centric Architecture Knowledge Management Environment) is a web-based tool built on top of the open source groupware platform Hipergate. It is aimed at providing knowledge management support for the software architecture process. It consists of four components; web-based user interface, knowledge management, search, and reporting. "Corresponding to these components, PAKME's features can be categorized into four AK management services: knowledge acquisition, knowledge maintenance, knowledge retrieval, and knowledge presentation [17]". PAKME's knowledge repository is logically divided into knowledge-based artifacts, generic knowledge, and project-based artifacts [23].

1.3. Problem statement

Despite the benefits of decision documentation, architecture decisions are almost never explicitly documented in the industry, but reside in the architect's mind as tacit knowledge or are only implicit in the models that the architect creates [4, 6]. This is partly due to the fact that there is currently no decision documentation approach or tool that is suitable for use in industrial projects. Several studies identified limitations of current tools, which makes them only suitable for particular tasks or under certain conditions [14, 17, 18]. Moreover, many existing decision documentation approaches do not integrate well with the rest of the architecture documentation (e.g. models, diagrams, views), as well as with the processes and tools that are used to design and document the architecture [5] . We will address the problem in the following way:

- We develop an extension to Sparx Systems' Enterprise Architect: a general-purpose model-driven, and extensible UML tool that can be used to create various types of models to support the architecting process. The resulting add-in should allow to document decisions in a user-friendly and efficient way.
- We conduct an exploratory case study to assess the status quo of architecture decision documentation, to identify expectations of the ideal documentation approach, and to evaluate the advantages and limitations of the developed tool.

[3] http://www.archium.net [Online accessed: 28/08/2013]

1.4. Approach

The project serves two purposes, the development of a decision documentation tool and to get further insights into the challenges of decision documentation in practice. Design science is a research framework that is optimal for these purposes since it serves to solve both practical and knowledge problems. It as an iterative research framework, which acknowledges that researchers "can produce scientific knowledge by designing useful things [24]". It constitutes three core activities: problem investigation, building, and evaluation.

In our case, the building activity is the implementation of the decision documentation tool that should address the aforementioned problems. Problem investigation is a knowledge question with which we seek to gain an understanding of the current status of decision documentation at our industry partner, and elicit their needs and expectations of the ideal decision documentation approach. The evaluation assesses to what extent the developed tool is suitable to solve the problem, and to get further insights to improve the tool. Therefore, we elicited requirements from practitioners and conducted an exploratory case study.

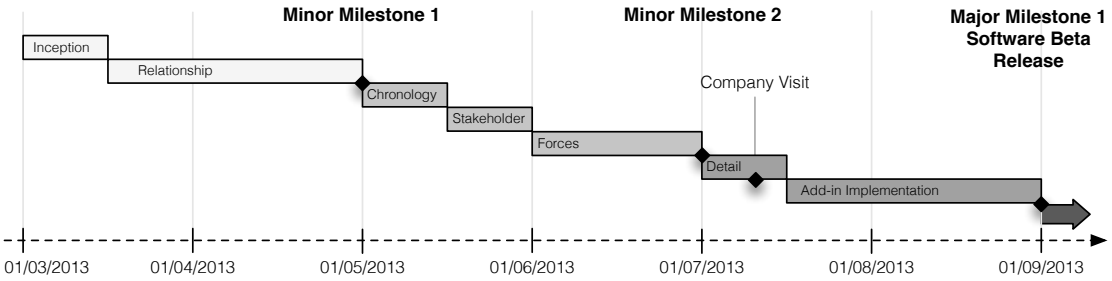


Figure 3: Project phases and milestones.

The time plan in Figure 3 illustrates the main phases and milestones of the project. Although the project is still in progress, we only show the phases relevant for the time-period of our thesis. The kick-off of the project took place at the beginning of March 2013. The purpose of the inception phase was to get familiar with the decision framework, related literature on architecture decision documentation, the development environment, and the Enterprise Architect (EA) API. A requirement for Minor Milestone 1 (May 1st 2013) was the design and implementation of the Relationship viewpoint.

As part of the next Minor Milestone 2 (July 1st 2013), the Chronology, the Stakeholder Involvement and the Forces viewpoint were designed and implemented.

The Detail viewpoint was designed and implemented during in next phase, in which the exploratory case study was also conducted . In the final phase until Major Milestone 1 (September 1st 2013), the results from the case-study were analyzed and several features like reporting were implemented.

2. Analysis, Design and Implementation

Software projects can be divided into three main activities, analysis, design and implementation. Analysis is concerned with getting an understanding of the project's environment, context and requirements. The problem domain is investigated in order to establish what the system should do, its essential and desired emergent properties, and the constraints on the system operation and the software development process [25]. This includes the definition of high-level use cases and key drivers as well as the elicitation and specification of requirements. In our case, the domain and the context are Enterprise Architect, ABB and the Decision Documentation Framework. Design is concerned with finding a solution that is suitable to address the requirements. For example, by decomposing the system into components, by choosing patterns, or by integrating existing technologies into the design. Eventually, the established design is implemented and evaluated. The activities are not performed sequentially but rather iteratively as it is not possible to get a complete understanding of the problem in the first place. In this project, we used an agile development approach that acknowledges the increasing understanding of the problem domain during the course of the project by organizing the development process into short cycles, each consisting of analysis, design and implementation.

However, this section is organized sequentially by first describing the domain of this project. After that the high-level use cases, requirements and key drivers are presented that lead to the design of the tool's architecture.

2.1. Domain

This project is part of an industry-research collaboration between the Software Engineering and Architecture group of the University of Groningen and ABB. The latter is a multinational corporation specialized in power and automation technologies. ABB employs around 145.000 people and operates in approximately 100 countries. ABB operates several corporate research centers around the world, which also includes research in the field of Software Engineering and Software Architecture.

The decision documentation tool is developed as an extension to Enterprise Architect, which is used by ABB modeling platform. EA is a general-purpose UML modeling tool developed by Sparx Systems. It can be used for architecture design, use case design, definition of scenarios, requirement specifications and for a multitude of other modeling purposes. EA is extensible via an Automation Interface[4] and via UML Profiles[5]. The first extension point provides the possibility to extend and automate the functionality of EA and to perform operations on the modeling objects, the so called EA Object Model. The EA Object model defines entities like elements, diagrams, and connectors

[4] http://www.sparxsystems.com/enterprise_architect_user_guide/10/automation_and_scripting/automation_interface.html [Online accessed: 28/8/2013]

[5] http://www.sparxsystems.com/enterprise_architect_user_guide/10/extending_uml_models/umlprofiles.html [Online accessed: 28/8/2013]

and their attributes and operations. The Automation Interface is event-based. For example, the extension is notified if elements are created or modified. The second option is to define a UML profile, which can be used to customize UML models for particular domains and platforms. The UML profile can define new element types, relationship types, and diagram types. In addition, EA provides the possibility to also customize diagram toolboxes and the project browser via UML profiles.

2.2. Use-cases

According to [26], use cases capture a contract between the stakeholders of a system about its behavior. The fundamental use cases of the add-in are written in text form and describe the most important functionality that the tool should exhibit. The textual form serves to communicate the goals of the add-in between the members of the project. At the beginning of the project, the following high-level use cases are envisioned for the add-in:

UC1 – Support the creation of decision views according to the five viewpoints.
This should be accomplished in a user-friendly and efficient way. Information that is specified in one of the views will be automatically propagated to the other views, so that no redundant information has to be provided by the user. Only the essential fields will be mandatory, while users will be able to select what information to fill in and what to skip. Emphasis will be given on keeping the user interface simple and reducing the learning curve.

UC2 – Import decision information from other sources.
A user-friendly wizard will allow to document decisions based on information imported from other sources like MS Word, Powerpoint, Excel, or Visio files. This will increase efficiency as information does not have to be re-inserted.

UC3 – Trace relationships between decisions and architecture design elements.
Such relationships can be established from both directions: decisions to design elements, and vice versa. This will apply both to direct and transitive relationships at configurable levels of depth. Examples of related design elements are components and connectors, deployment diagrams, or other types of models that can be created using Enterprise Architect.

UC4 – Export decision documentation.
The created decision views and their relationships with other design elements can be exported for use in other software tools, in order to create reports. The available formats (e.g. MS Office Word, or PDF) will be agreed upon with ABB. Reports will be generated based on templates; for example in the case of MS Word, a report template can be used that contains decisions views, their traces to other design elements and placeholders for explanatory text.

The aforementioned use cases provide a good overview of the main goals of the add-in and a solid ground on which to base the requirement elicitation process.

2.3. Requirements

Requirements establish what the system should do, its essential and desired emergent properties, and the constraints on the system operation and the software development process [25]. Requirements are discovered during requirements analysis and elicitation, in which the developer involve stakeholders to understand their needs and expectations of the system. The objective of the requirements engineering process is a Software Requirements Specification (SRS), which is an official statement of what the system developers should implement [25].

The first step of the analysis was to familiarize with the Decision Viewpoint Framework. Therefore, the framework has been applied to retrospectively document architecture decisions for a recent software project. The produced documentation states architectural-significant requirements, forces, and stakeholders. In total, we documented 19 decisions in the detail view and established the chronology and relationship view for the project. The document can be found in Appendix D. Based on the insights from this activity, a prototype has been implemented. In the next step, in-depth requirements interviews with potential end-user of the tool were conducted. The interviews were semi-structured as this allows to discuss the main use-cases but also uncover innovative requirements. The interview guide was on the high level use cases and the insights from the prototype. The complete interview guideline used for the requirements elicitation interviews can be found in Appendix B.1. Prior to the interviews, the participants had the opportunity to try-out the functional prototype of the add-in as this helps to identify and improve any emerging usability issues as early as possible in the development phase. In total three interviews were conducted and another architect filled out the interview guide offline. The interviews were recorded, transcribed and allayed.

The result of the interviews was a preliminary set of 86 functional and quality requirements. The requirements were preliminary prioritized by us to exclude those that are not feasible due to technical limitations or because they are beyond the scope of this project. The initial prioritization was validated by the stakeholder based on a MoSCoW[6] scheme. In total, 31 must, 16 should, 23 could and 14 won't requirements made it into the specification. The SRS has been written in conformance with IEEE 830-1998 standard for requirements specifications and can be found in Appendix C.

2.4. Key Drivers

The selection of key drivers is the first major architecture decision of the add-in as the "shape" the architecture. The initial project proposal mentions two key drivers, usability and efficiency. We consider maintainability equally important for this project. The selection of these key drivers is based on the expectation that the result is a usable and efficient tool to document and integrate architecture decisions in architecture documentation that provides value to practitioners.

[6] Must, Should, Could, Won't

During the requirements elicitation interviews, additional quality attributes came up, such as portability and security. However, their importance was not considered high-enough to be one of the key drivers.

Usability

According to Bass et al., usability or user-friendliness is concerned with how easy it is for the user to accomplish a desired task and the kind of support the system provides to the user [12]. This can be achieved by seamlessly integrating the tool into EA, following its recommended guidelines. For example adhering to the specified design concepts, such as toolboxes, and diagrams. To further increase usability, we provide an intuitive way for users to incorporate decision views into regular architecture projects. Special attention is given to more complicated features, such as generating the Chronology viewpoint, or incorporating the Forces viewpoint as a table into EA.

Efficiency

Efficiency is often found under the umbrella of performance. In the case of the add-in, the role of efficiency is twofold. The first part concerns the quick response to users' requests [12]. The second part is about reducing the time required for documenting architecture decisions. This is the reason that particular emphasis is given in automating some of the functionality of the add-in, such as the generation of the Chronology viewpoint. This is also a trade-off between efficiency and usability, as complicated features might result in usability issues.

Maintainability

The source code should be structured in such a way to increase its maintainability and reduce the learning curve of new project members. This is crucial for this project, as the team responsible for the implementation will change. New members will need to join the project, so they should be able to understand the code and be productive with as soon as possible.

2.5. Architecture of the Add-in

Every system has an architecture even this EA add-in. The add-in extends the EA Automation Interface and provides a custom UML profile. The UML profile specifies the framework's model element such as the decision entities, decision states, relationship and viewpoints (cf. Figure 2). The profile is encapsulated in a MDG technology file, which seamlessly plugs into Enterprise Architect to provide additional modeling resources. This section presents three views on the architecture and lists the most important architecture decisions.

2.5.1. Logical View

The logical view primarily supports the functional requirements, the things that the system should provide in terms of services to its users [27]. We decompose the system as a set of key components and connectors. According to Kruchten, this "decomposition is not only for the sake of functional analysis, but also serves to identify common mechanisms and design elements across the various parts of the system."

As illustrated in Figure 4, the system is logically structured into four components: MainApplication, DecisionViewpoints, CustomViews and the EAFacade. Interop.EA and OpenXML are third-party libraries.

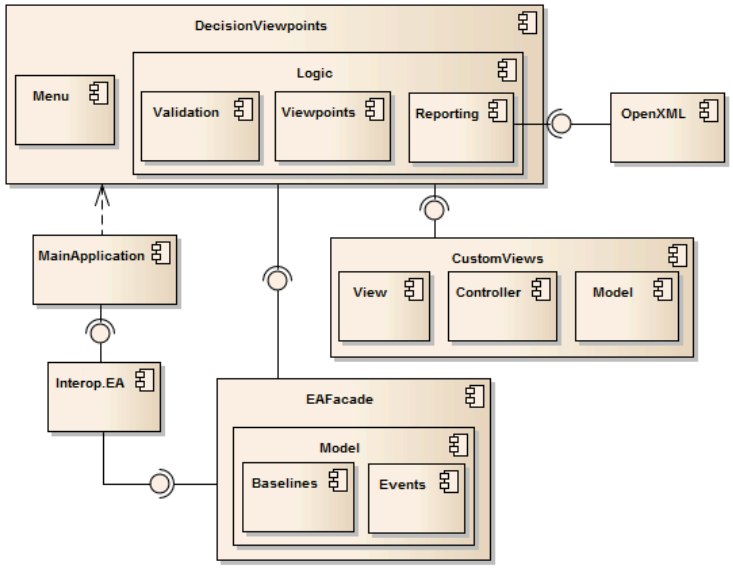


Figure 4: The component view of the add-in.

MainApplication
: Is the client-side implementation of the Automation Interface contract. It is implemented using the Application Facade Pattern to dispatch the events to the appropriate handlers.

DecisionViewpoints
: This component contains the main logic of the add-in. It follows a design rule to not directly use the Interop.EA but to access all EA-related functionality via the EAFacade.

  Viewpoints
  : Viewpoints contains the logic required to create and maintain the five viewpoints. For example, the automatic generation of the Chronology view.

Validation
: The add-in uses the model validation feature of EA to allow users to validate created views. The viewpoint constraints that are defined in the documentation framework [5] are structured as rules. For example, a rule could be that a decision of state *idea* is not allowed to have any relationship to any other decision. The models are then validated by EA against all these rules and related warnings are presented to users.

Reporting
: Reporting is responsible for generating reports. Three kind of document types are supported Word, Excel, and PowerPoint. It accesses the OpenXML API, which defines a set of XML schemas for representing spreadsheets, charts, presentations, and word processing documents.

Menu
: Provides the structure and the logic of the context menu of the add-in. It is responsible for handling the events which are raised when users select options from the context menu, for instance to generate a Chronology view.

CustomViews
: This components implements a large part of the logic behind the Detail and the Forces viewpoints. It is structured using the MVC pattern [28] and it exposes an interface to DecisionViewpoints. Part of the logic implemented in this component is to store data from the Forces and Detail views, like the ratings from the Forces view, and fields of the Detail view, like argumentation.

Interop.EA
: Interop.EA provides a unified interface to the EA API.

MainApplication
: The MainApplication implements the interface of the EA API . It extends the EAEventAdapter class and delegates events to its listeners, for instance events related to model validator, or repository events (element creation, deletion, or modification).

EAFacade
: It uses the Facade pattern [29] to create a unified access to the EA API. It defines a higher-level interface that makes the EA API subsystem easier to use. It also provides low-level helper functions that a frequently used by the add-in.

Events
: The interface to the EA API events, i.e. when an element is created, deleted, or modified. This components does not contain any logic but interface definitions.

Baselines
: Provides access to the XML-based baseline interface of the EA API. It is responsible for creating and retrieving baselines as well as parsing the results of baseline comparisons.

2.5.2. Implementation View

The implementation view focuses on the actual software module organization on the software development environment [27]. The implementation architecture of the add-in focuses on the ease of development, software management, reuse of software components, and on the constraints imposed by the toolset. Typical stakeholders for this view are new project members. It provides a good overview of the structure of the add-in's codebase. We briefly discuss the subsystems and their relation to the aforementioned requirements.

Add-in

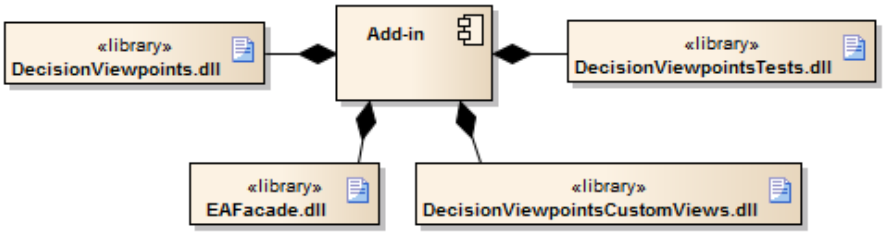


Figure 5: The high level implementation view of the add-in.

Integration of the tool into EA imposes the constraint of using either C# or Visual Basic as a programming language. C# is used for this project as requested by ABB. Microsoft Visual Studio Ultimate 2012 is used as development environment. Furthermore, ReSharper[7] has been used to extend the features of VisualStudio. ReSharper has been mostly used to establish a uniform coding guideline.

Figure 5 shows the high-level decomposition of the add-in. The functionality is decomposed into three subsystems, DecisionViewpoints.dll, EAFacede.dll, and DecisionViewpointsCustomViews.dll. The reason for such a decomposition is a clear separation of concerns. DecisionViewpointsCustomViews.dll is responsible for the implementation of Forces and Detail views, EAFacade.dll provides a unified access to the EA API, and DecionViewpoints.dll contains the main entry point to the add-in and the logic behind the views. Each subsystem has a well-defined responsibility.

DecisionViewpoints

The DecisionViewpoints library provides the main entry point of the add-in. It implements the logic for the viewpoints (Handlers) and other features, such as the context menu and reporting. Figure 6 shows the implementation view of the DecisionViewpoints library. The MainApplication implements the main class of the add-in that is used by EA. The MainApplication extends the EAEventAdapter class and delegates events to its

[7] http://www.jetbrains.com/resharper/ [Online accessed: 3/9/2013]

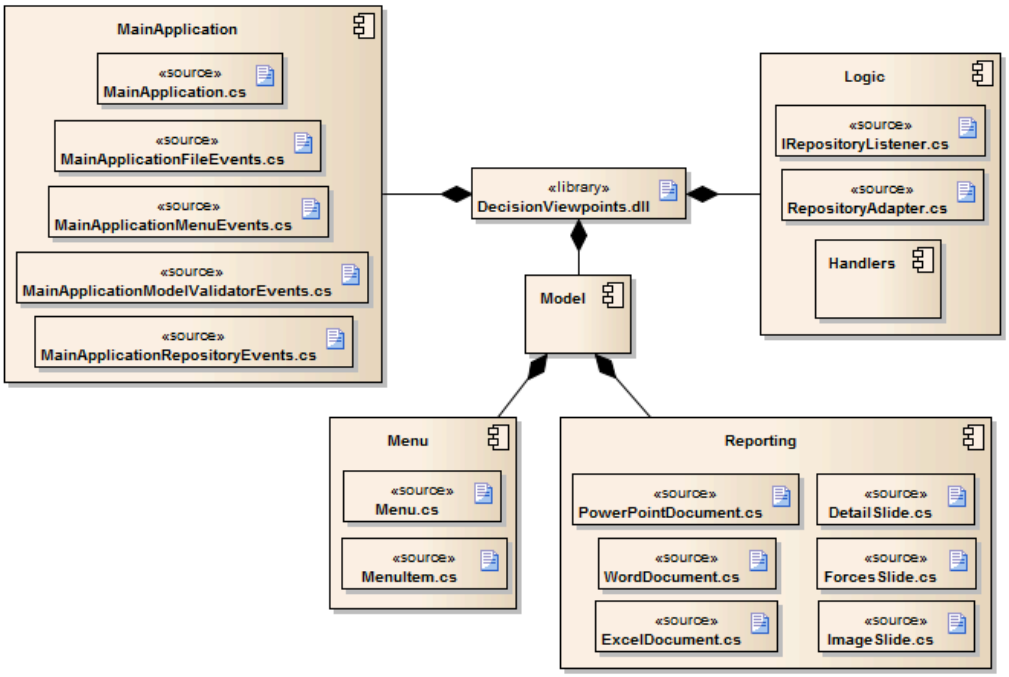


Figure 6: DecisionViewpoints library.

listeners by using the Observer design pattern [29]. It is split into five parts, each part is responsible to delegate specific events. The Menu subsystem is responsible to handle user interaction with the context menu of the add-in. It uses the Composite design pattern [29]. In our case, the root menu consists of menu items and menu items consist of child menu items . The Reporting subsystem implements the functionality that allows users to generate reports. The Factory design pattern [29] is used to generate different kinds of reports. The ReportFactory class is contained in the Handlers subsystem.

DecisionViewpoints - Handlers

Figure 7 shows the source files and subsystems that constitute the Handlers subsystem. The viewpoint handlers implement the logic behind the viewpoints. For example, when a user double-clicks on a decision element, the Detail view opens. The viewpoints handlers are split in different classes resulting in a clear separation of concerns.

The Menu subsystem contains the menu handler, which serves menu events. For example, when a user selects the option to generate the Chronology view or to create a report. The functionality to follow traces is also implemented in the Menu subsystem. The ReportFactory class is responsible for creating generic document objects and remove dependencies on concrete document classes. The Validation subsystem implements the model validation feature. The constraints are implemented as rules.

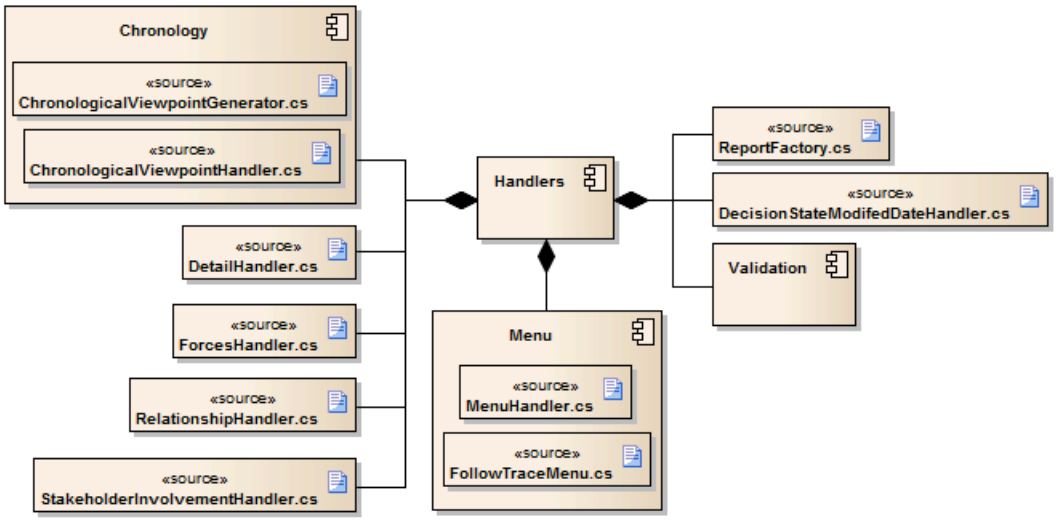


Figure 7: Handlers within the DecisionViewpoints library.

EAFacade

The main purpose of the EAFacade is to provide a unified access to the EA API. It is an implementation of the Facade design pattern [29]. Figure 8 shows the source files and subsystems within the EAFacade library. The Facade wraps model elements of the EA API (connectors, elements, packages) and provides access to their attributes and operations in a more easy and convenient way. These models are contained in the EAModels subsystem. The EAFacade also includes the extended UML Profile of the add-in (MDG Technology file) DecisionViewpoints.xml, and the two template files DefaultDecisionViewTemplate.xml and EmptyDecisionViewTemplate.xml. The Baselines subsystem implements the models that represent baselines. The Events subsystem consists of classes responsible for providing an interface to EA Events. The main class of the add-in extends the EAEventAdapter class, which uses the Adapter design pattern [29].

DecisionViewpointsCustomViews

The CustomViews library (cf. Figure 9) uses the Model-View-Controller architectural pattern [28] to structure the Forces and Detail views. There is no equivalent EA diagram that could be used to implement both the Detail and Forces viewpoint since they require customized user interface controls, such as textfields or tables. The ForcesModel and Decision are the models that encapsulate relevant information. For instance, Decision encapsulates the argumentation and issue behind a decision, while ForcesModel encapsulates the forces, alternatives, and ratings associated with a particular decision.

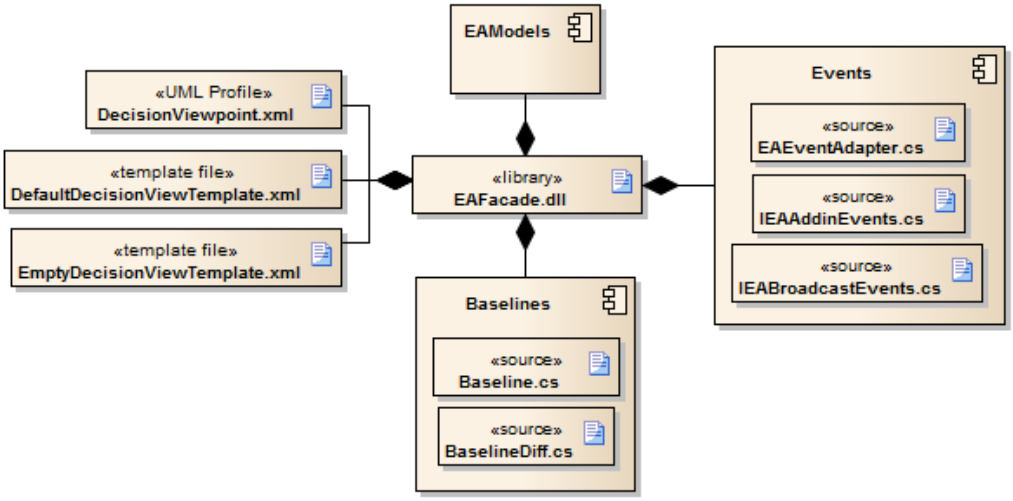


Figure 8: EAFacade library.

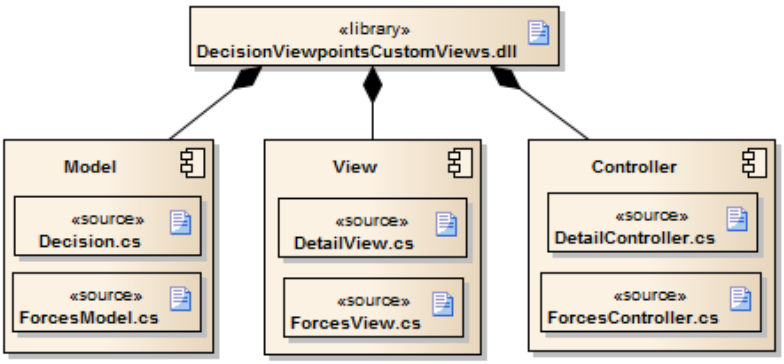


Figure 9: DecisionViewpointsCustomViews library.

2.5.3. Deployment View

The add-in is deployed as a set of DLLs[8]. One of the DLLs contains the main entry point of the add-in, the main class. This DLL is the loaded by EA. The MDG technology file is included into the DLL as a resource file so that the add-in can be easily deployed in a company-wide network infrastructure. Figure 10 illustrates the add-in architecture of EA. When EA starts, it looks up the Windows registry for defined add-ins. In the next step, EA queries the COM codebase entries in the registry for the location of the DLL and loads it. From now on, the add-in is registered and is ready to receive events. Two classes of events are fired by EA, add-in events related to the program life-cycle of EA and broadcast events about the open EA project file.

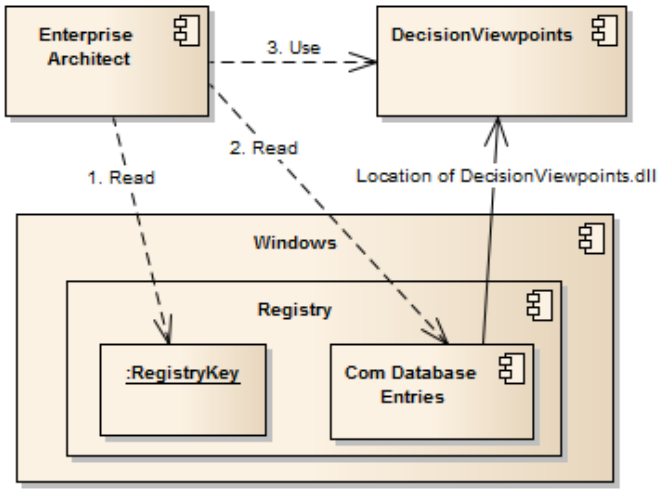


Figure 10: Add-in architecture of Enterprise Architect[9].

The compilation of the add-in produces three DLLs: the DecisionViewpoints.dll, the CustomViews.dll, and the EAFacade.dll. All dlls of the add-in are deployed on the client machine. The deployment view, also sometimes referred to as physical view, takes the system's non-functional requirements [27] into account. The main non-functional requirements of the add-in are usability, efficiency, and maintainability (cf. Section 2.4). Portability and security were also discussed during requirements elicitation. Portability is addressed, essentially by not using any deprecated APIs, as all external APIs are maintained and are up-to-date. Moreover, the add-in supports Windows 7 and at least version 10 of EA.

[8] Dynamic-link library is a shared library in Windows-based operating systems.
[9] Reproduced from http://bellekens.com/2011/01/29/tutorial [Online accessed: 28/08/2013]

Security is addressed by integrating with the standard security features of EA. The deployment view does not address any usability issues. To increase maintainability, the main functions of the add-in are separated according to their concerns, as illustrated in Figure 11 and as described in the implementation view.

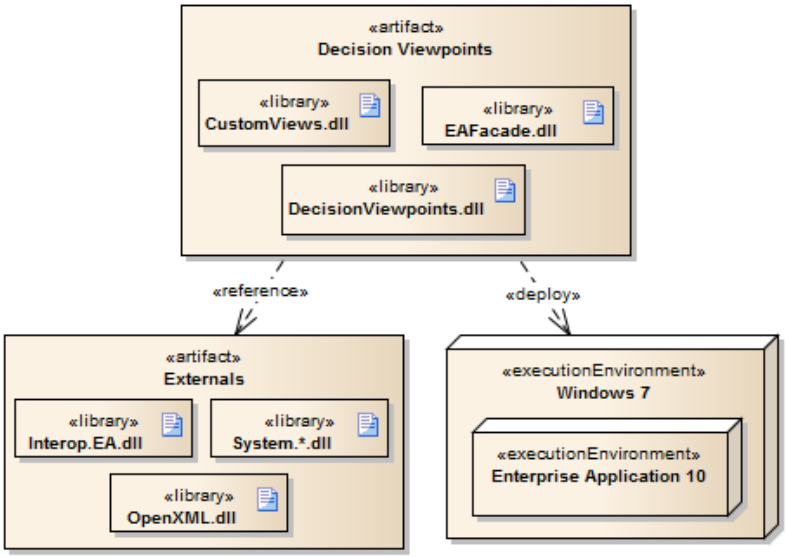


Figure 11: The deployment view of the add-in.

2.5.4. Design Decisions

Architecture can be seen as the sum of decisions that were made during the architecting process. We followed this perspective on architecture by using our own tool to document the decisions that we made during the development process. In this section we present the fundamental architecture decisions, the underlying issue and forces, the considered alternatives, the rationale and an evaluation of this outcome of the decision.

Storage for decision state changes

Issue: As stated in the requirements, it should be possible to create the Chronology viewpoints manually but also to generate it automatically. Therefore, the add-in must be able to identify all state changes of a decision and the time and date of the change, for example when a decision evolves from *idea* to *decided*. However, EA architect does not automatically maintain a change log and thus possibility to store state changes has to be found.

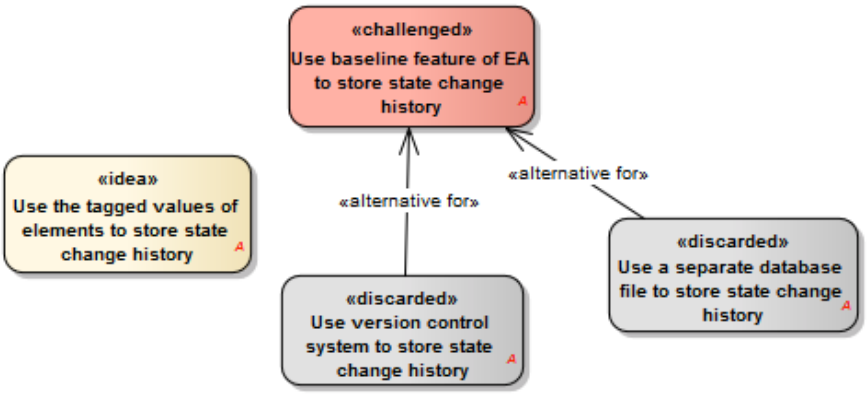


Figure 12: Storing decision state changes.

Forces: Several forces influence the decision. The solution should be integrated into EA and it should be easy to share the state changes along with the EA project file that contains the decision model. Furthermore, the storage has to be persistent and performant.

Alternatives: Three alternatives were identified that are able to address this issue. The first one is to the baseline feature of EA. The second one is to use a version control system to capture the changes in the model. The third alternative is to store the data in a separate database file (e.g. sqlite).

Argumentation: It has been decided to use the baseline feature of EA. Baselines capture snapshots of EA models. By reading and comparing these baselines it is possible to extract changes in the model and to generate the chronology of state changes. A problem with this solution is determine when to capture a baseline of a model in order to not capture accidental state change or to omit important state changes. Three option were implemented; a manual invocation by the user through a context menu of the add-in; whenever the user closes an EA project file; or on every state change. The user can decide which option is most suitable for him.

The version control alternative was discarded because an evaluation of the technology showed that the feature is not reliable. Furthermore, not every user is familiar with the version control feature of EA. It would considerably impede the usability of the add-in to enforce users to use the version control feature. The reason to discard the use if a separate database was that it would require additional setup and thus would negatively affect the integration of the add-in into the platform.

Evaluation: The baseline feature is suitable to capture state changes, however several challenges were identified. This includes a reduced maintainability of the code because of the complex logic to identify state changes and to generate the chronology. Moreover, while reading baselines EA opens and closes a dialog for each stored baseline. These dialogs cannot be suppressed and it turned out to be confusing for the users, impeding

the usability of the add-in. A third issue is that a lot of overhead is stored along with the state changes that is not necessary. For these reasons the decision to use baselines was challenged as shown in Figure 12. A new alternative was proposed to store state changes as tagged values[10] within the decision.

Strategy for creating a Forces view

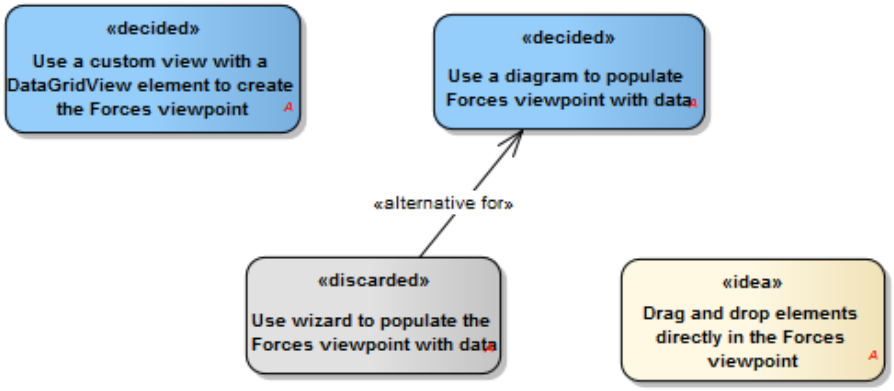


Figure 13: Creating a Forces view.

Issue: The Forces view is implemented as custom view because EA does not provide the ability to create tables as diagrams[11]. The custom view contains a data grid view, which is used as the Forces-Alternative matrix. No other alternatives were considered for this decision, as we were quite confident with the custom views solution. However, since Forces viewpoint consists of multiple elements, the issue was how the user can create and populate the Forces viewpoint with data.

Forces: Usability was the main motivation behind this decision. It should intuitive for user to populate Forces views with data.

Alternatives: There were two alternatives. The first one was to let users specify the data through a wizard-dialog. The second one was to use a diagram in which users could drag and drop elements (requirements, concerns, decisions) and then the viewpoint is generated out of the elements that were added to the diagram.

Argumentation: We decided to use the second approach, to to let the user drag and drop elements on a diagram. The Forces viewpoint is automatically updated when the diagram is updated. The reason to use this approach is because EA users are quite familiar with creating diagrams. Moreover, the fact that a Forces viewpoint has an associated diagram made it also easier for us to relate specific elements with each Forces

[10] A key-value datastore within each EA model element.

[11] http://www.sparxsystems.com/resources/developers/autint_vb_custom_view.html [Online accessed: 01/09/2013

viewpoint and save the problem of where to store the data. We discarded the use of a wizard as users would need to go through various steps to specify the data and this would be too time-consuming. Furthermore, the wizard is requires more effort to implement.

Evaluation: During the case study one of the architects had the idea to directly being able to drag and drop elements from the Project Browser onto the forces-matrix. However, the idea is not evaluated and implemented yet. Figure 13 shows the relationship view of this decision. In general, the current way of creating a Forces view is convenient and intuitive.

Strategy and implementation of reporting

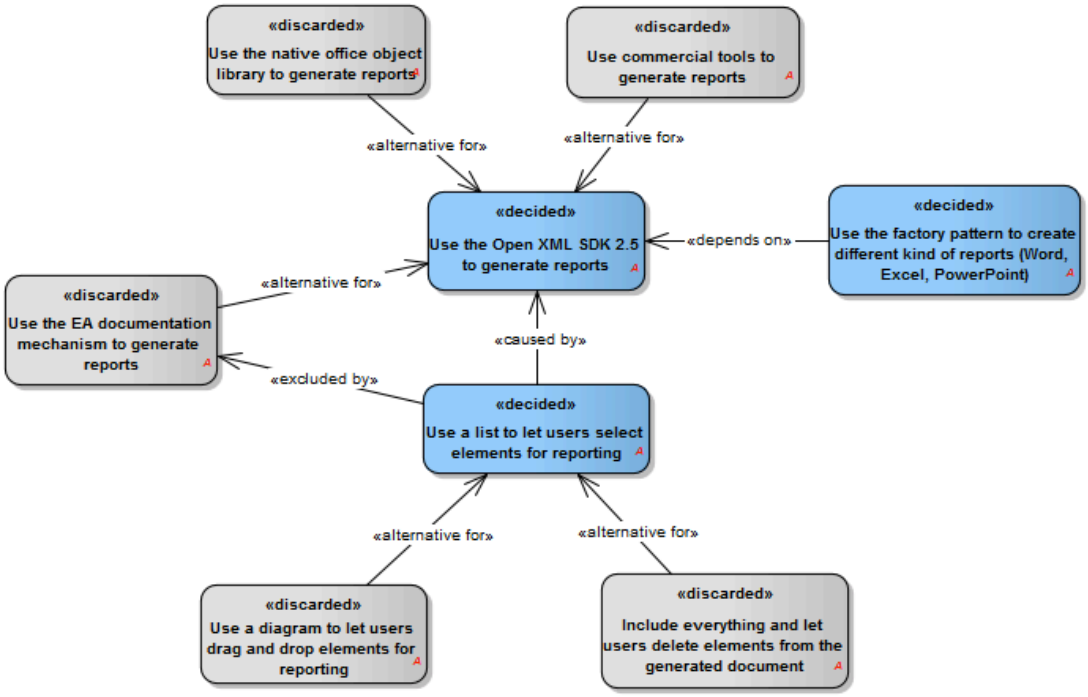


Figure 14: Reporting mechanism.

Issue: Generating reports is one of the high level use cases.

Forces: The goal is to create a user-friendly reporting mechanism. The requirement is that users should be able to export decision documentation in Word, Excel, PowerPoint, and PDF in a user-friendly way. It should also provide the development team enough flexibility to create those kind of documents and manipulate their structure, for example to be able to include the Detail viewpoint of a decision as a table in a Word document.

Alternatives: Four alternatives are considered, the native office object library, a commercial solution, the EA documentation mechanism, or the Open XML SDK 2.5.

Argumentation: The decision is to use the Open XML SDK 2.5. The Relationship viewpoint of this decision is shown in Figure 14. Open XML SDK 2.5 provides a structured way to create and manipulate office documents (Word, Excel, PowerPoint). The idea of using the native office object library is discarded, as it is too low-level library and it would increase too much the development time and the code complexity. Although commercial tools would decrease by far the development time, this alternative is discarded as it is not a free solution. The EA documentation mechanism lets you specify which models should be included in the report by adding them in a diagram. Although this seems as an intuitive and user-friendly way, it is discarded it. The reason is that it is not certain that users would be familiar with this option of EA. Furthermore, it does not provide enough flexibility, for example it is not able to specify how the documentation would look like. To have the flexibility of generating different kind of documents and increase the maintainability of the code, it is decided to structure the reporting functionality using the Factory pattern [29]. This decision depends on the decision to use the Open XML SDK 2.5. For example, if EA documentation mechanism would be used, then the decision about the Factory pattern would not have meaning any more.

The users should also be able to select which decision viewpoints should be included in the report. This increases the usability of the add-in. Regarding this decision, there are three alternatives. To use a list to let users specify elements, use a diagram and let users specify elements by dragging and dropping them in the diagram, or include everything in the report and let users delete what they do not need from the generated document.

The last option would decrease the complexity and the development effort, however it is definitely not a usable solution. Using a diagram seems like an intuitive way, but it would increase the complexity and the development effort. Therefore, it is decided to use a list. This alternative requires some development effort but it provides an easy and familiar solution to users.

Evaluation: The current implementation of the reporting mechanism does not provide enough flexibility to users, as they can select a specific set of decision views to include in the report. Generating PDF documents is also not supported currently.

Abstraction from EA API

Issue: The add-in depends on the EA API. While experimenting with the API, one issue is identified. The issue is that accessing the API is not so intuitive and it is rather difficult to structure the code.

Forces: The solution should be preferably free to use. Though commercial solutions can be considered. The solution should be easy to customize to fit the needs of the project. Moreover, the solution should not require a lot of implementation effort, or increase the complexity of the code.

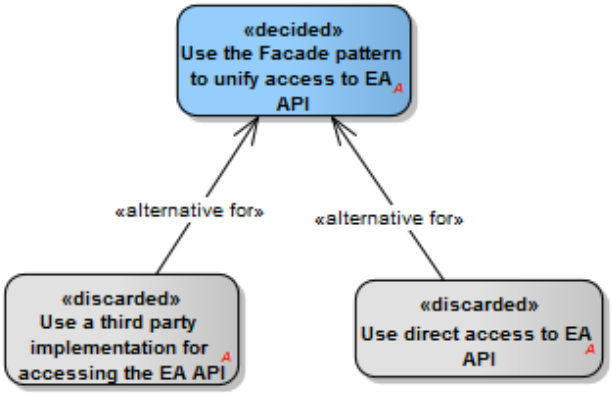


Figure 15: Accessing the Enterprise Architect API.

Alternatives: Direct access to the API is a possible solution, but this causes a lot of maintainability problems with the code. Two alternative solutions are considered. To use a third party implementation to access the API, or to create our own implementation for accessing the API.

Argumentation: It is decided to use the Facade pattern [29] to unify the access to the EA API (cf. Figure 15). This increases by far the maintainability of the code by simplifying the access to the EA API. It provides a simplified interface to the overall functionality of EA API. Though more implementation effort is required in order to create an interface from scratch. The third party implementation solutions are discarded as they do not fully cover the requirements and additional modifications would still be required. Furthermore, it is not certain that the third party implementations are maintained.

Evaluation: This solutions simplifies a lot the interaction with the EA API.

Storage for decision rationale

Issue: A native EA element is used to model the decisions. This provides the possibility to re-use some of the EA model element data, such as the name for the decision name and the stereotype for the decision state. However, it is important to let users provide additional information for some decisions, such as argumentation and problem/issue. The challenge is to find a place to store this additional information.

Forces: The implementation should be easy and it should not increase the code complexity. The solution should also be flexible enough to allow theoretically infinite amount of data to be stored. For example, it is not known beforehand how large would be the text for the argumentation.

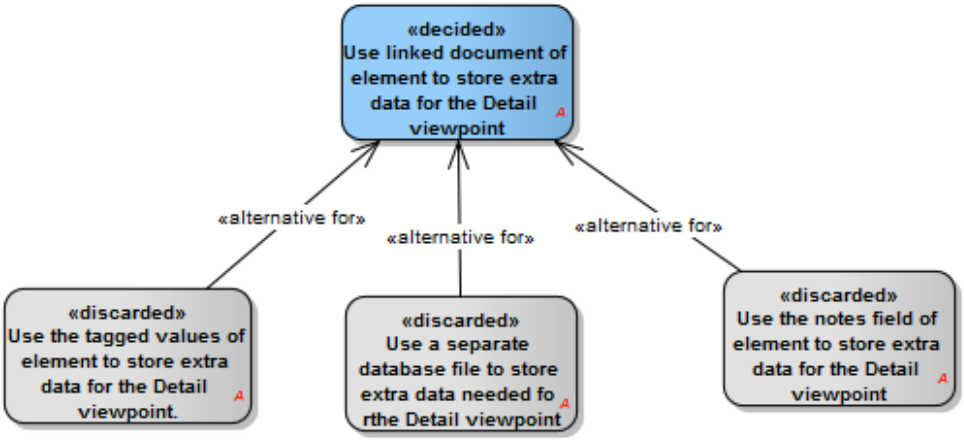


Figure 16: Storing the rationale contained in Detail view.

Alternatives: Four alternatives are considered, to use the tagged values of the EA model elements, use a separate database file, store the data in the notes field of the model elements, or use another mechanism of EA and store the data in a document linked with a model element, namely linked document.

Argumentation: The use of linked document is decided to store the additional information needed for the Detail viewpoint. The implementation is quite simple and theoretically unlimited amount of data can be stored. This solution is of course valid as long as users do not use the linked documents of the EA model elements. Figure 16 depicts the Relationship viewpoint of this decision.

The alternative to use tagged values is discarded, as tagged values are only limited to 255 characters and this might not be enough for users to provide for example argumentation. The alternative to use a separate database file is also discarded as it would pose many synchronization issues. If multiple users are working on a single project, they would need to synchronize the database file. This introduces too much additional hassle for them. The option to use the notes field of the model elements is not flexible enough as there is a high probability that it is used for other purposes, and this would increase the complexity of the implementation.

Evaluation: So far the use of linked documents to store decision rationale is proved effective.

Strategy for auto-generating viewpoints

Issue: A requirement is that users shall be able to generate the Chronology viewpoint, apart from being able to create it manually. The interviewees said that the time needed to manually create the Chronology viewpoint is not seen as beneficial and they should be able to generate it, to avoid creating it manually.

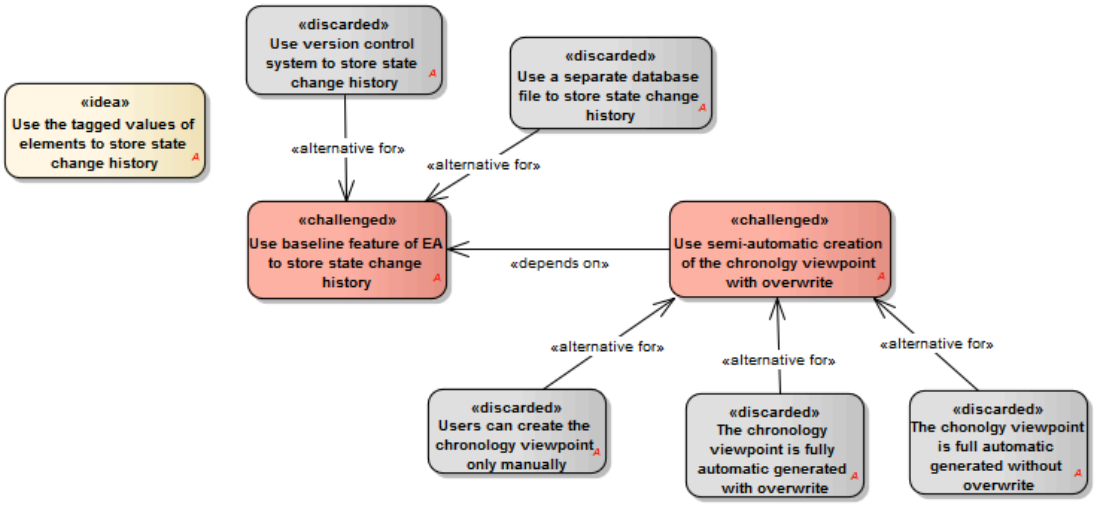


Figure 17: Generating the Chronology view.

Forces: The solution should not impede the usability of the add-in. It shall be easy for users to generate Chronology viewpoints. Moreover, implementing any of the solutions should not increase the code complexity.

Alternatives: To address this requirement, three alternatives are considered. The Chronology viewpoint could be generated in a semi-automatic way, where the users could select an option from the context menu of the add-in to generate it. Then by reading and comparing the baselines to get the history of state changes, the viewpoint could be created. However, the new generation would overwrite any previous changes. Another option is to generate the Chronology viewpoint automatically in every state change and overwriting any previous changes. The third option is to generate the Chronology viewpoint fully automatic in every decision state change without overwriting previous changes. Of course there is also the option to only let the users create the Chronology viewpoint manually.

Argumentation: It is decided to use the semi-automatic way. Figure 17 shows the Relationship view regarding this decision. Both of the automatic ways are discarded, because this would increase by far the complexity of the implementation. Moreover, these two options would not provide enough flexibility, for example when there is a state change, which Chronology view should be generated (a project can contain multiple Chronology views). The semi-automatic way allows some flexibility as users can select which Chronology view to generate. At the same time it does not increase at the same amount the complexity of the implementation as the fully automatic options do.

Evaluation: During the case study a lot of comments were received on the generation of the Chronology viewpoint, with respect to maintenance issues. Therefore, the decision is challenged and needs to be evaluated again, and maybe come up with new solutions.

3. The Add-in

In this section, the developed tool is presented and the most important functions are explained. This includes the integration of the extension into the EA platform, the five different views, the traceability of decisions to other elements, the validation of the views and the possibility to generate reports. The functionality is illustrated by a decision, which was made during this project. The underlying problem of the decision was to find a way to store state changes of decisions in order to be able to generate the chronology viewpoint. The decision is discussed in more detail in Section 2.5.4.

3.1. Integration into Enterprise Architect

Extensions should be seamlessly integrated into the existing applications environment by picking up predominant usage patterns and by being consistently integrated into the existing user interface. The same applies to our add-in. Hence, an essential design objectives was that it should feel like a natural part of EA in order to increase usability and to minimize the learning curve. The add-in is built-in into the default building blocks of the EA user interface, such as the toolbox, diagrams, or project browser.

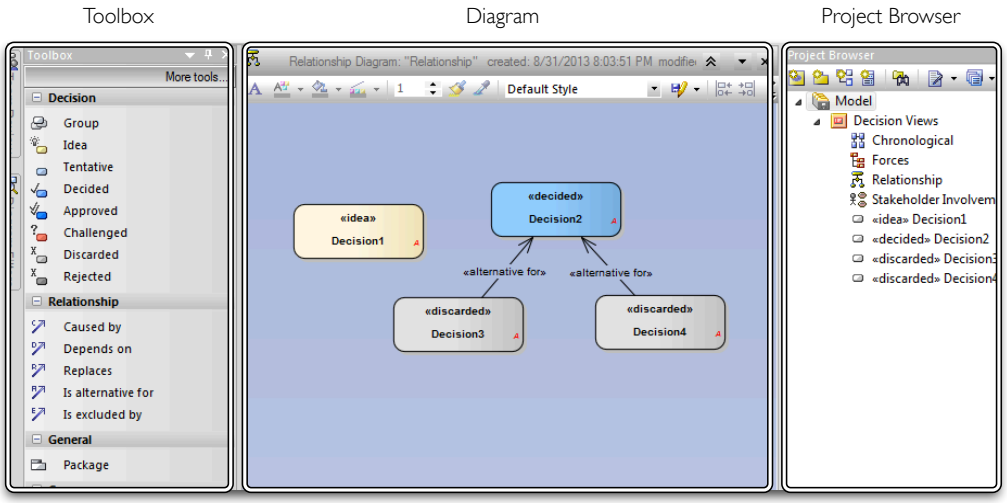


Figure 18: The user interface of Enterprise Architect.

Figure 18 shows the fundamental building blocks of EA's user interface. The toolbox shows available elements that can be dragged and dropped onto the diagram. The selection of elements in the toolbox depends on the diagram type, e.g. class, component, or state. The diagram in the center is the workspace to model and to layout views. The project browser shows the project structure and all concrete elements that have been created for a particular project, e.g. packages, elements, and diagrams.

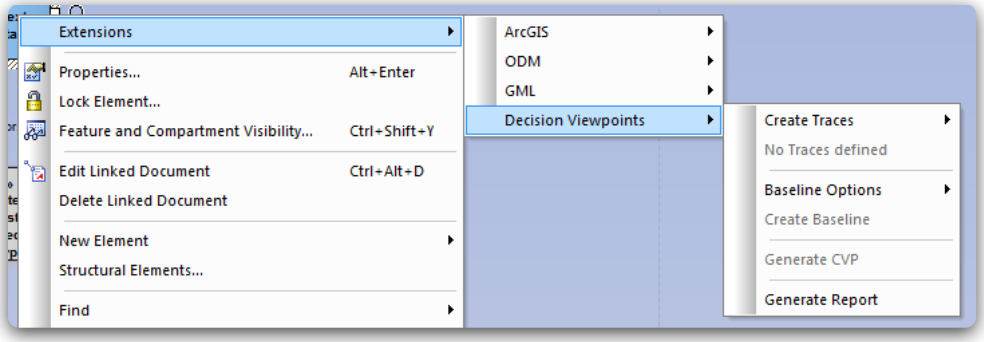


Figure 19: The context menu of the add-in.

EA add-ins have the possibility to provide additional functionality via a context menu. Figure 19 shows the context menu of our add-in from where the user can access additional functions like creating and following traces or generating viewpoints and reports. Additionally, they can modify, when and how the add-in captures state changes of decisions. The context menu of the add-in is context sensitive, which means that certain menu items are only available if a particular type of element has been selected. For example, the option to generate the chronology view is only available if a diagram of the type chronology is selected in the project browser.

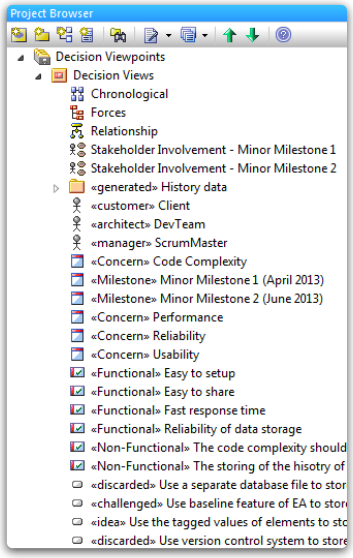


Figure 20: The project browser showing elements of the add-in .

By default all views and decisions are contained within a special package, called the Decision Views package. The Decision Views package accommodates four diagrams, one for each viewpoint: Relationship, Chronology, Forces, and Stakeholder Involvement. The project structure of the example is illustrated in Figure 20. The Decision Views package is used as a point of reference within the project structure. Amongst others, this is necessary to identify all decisions in order to generate a Chronology view. It should be noted that the user is free to choose its own structure and choice of viewpoints. For instance, it is possible to have multiple Decisions Views packages per project.

The decision element is the most important entity within the add-in as it represents the architectural knowledge, i.e. the problem, the justification, the evolution. As described by the framework, a decision can have different states depending on the current phase of its life cycle. Each state is associated with a different color to get a faster overview of the overall state of the decisions. The colors were selected carefully to express a semantic meaning. For example, challenged decisions are indicated in red to emphasize that this decision represents an unsolved problem and requires further discussion. Discarded or rejected decisions are displayed in an unobtrusive grey, while ideas resemble the bright yellow of a light bulb, which are a common metaphor for a sudden inspiration. Tentative, decided and approved decisions are shown in a gradient from light to dark blue to illustrate the increasing certainty of the decision.

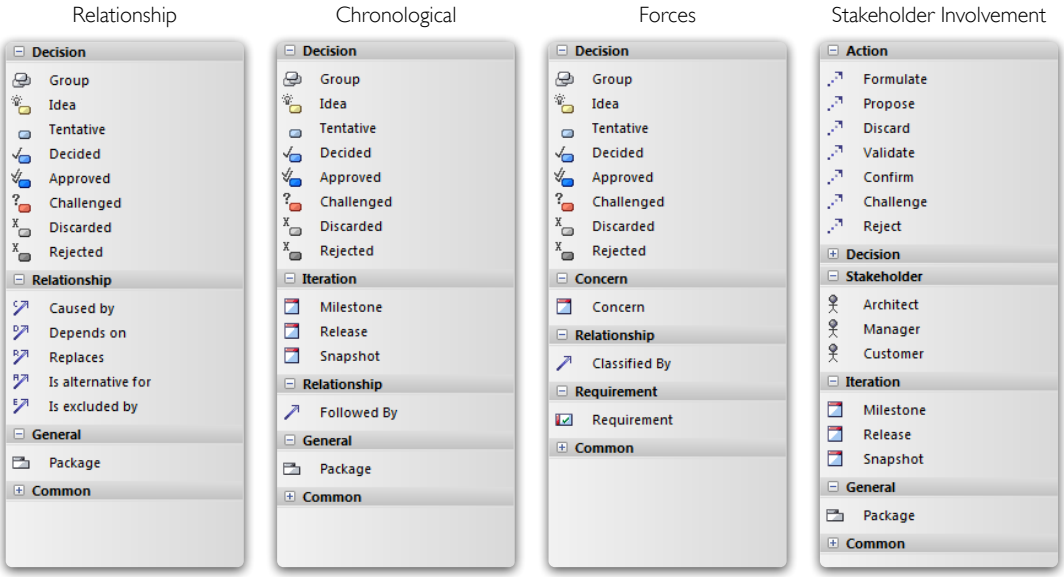


Figure 21: Viewpoint-specific toolboxes.

Each viewpoint has its own toolbox that contains only the elements relevant to create a particular view. The four different toolboxes of the add-in are depicted in Figure 21. The common elements contained in all toolboxes are decisions. In addition to these, the toolbox for the Relationship viewpoint provides connectors relevant for creating relationship between decisions. In contrast, the toolbox for the Chronology viewpoint contains elements to indicate iterations within the software process. The Forces' toolbox provides a concern and a requirement element. The Stakeholder's toolbox lists common stakeholders and the actions that they can perform on decisions. Each element within a toolbox has its own distinguishable icon. For example, the different decision states resemble the associated color.

3.2. Decision Views

This section shows the concrete views which have been created to document the previously mentioned decision. Although the decision was made in an iterative manner, the section focusses on the final outcome of the decision.

3.2.1. Detail

The Detail viewpoint is the default dialog that is shown when a decision has been selected. Figure 22 shows the Detail view of the decision. The fields *Related Decisions* and *History* are automatically filled out based on the information stored in the Relationship view (cf. Section 3.2.2) and the Stakeholder Involvement view (cf. Section 3.2.5). These fields cannot be edited by the user.

The *Problem/Issue* describes the underlying issue that the decision tried to solve. The *Decision* field describes the selected solution and the *Alternatives* field specifies the considered alternatives. The argumentation is one of the most important fields. In this example, the *Arguments* field describes the rationale that supports the *decision*. It contains the reasons of why the *alternatives* were discarded. Since the fields are rich text fields they provide the possibility to include links to websites or documents, e.g. links to requirement specifications or links to product pages of tool vendors.

The *Related Requirements* field lists the requirements that are used in the Forces viewpoint (cf. Section 3.2.4). This is not compulsory, but it is helpful to have most of the information aggregated in the Detail viewpoint.

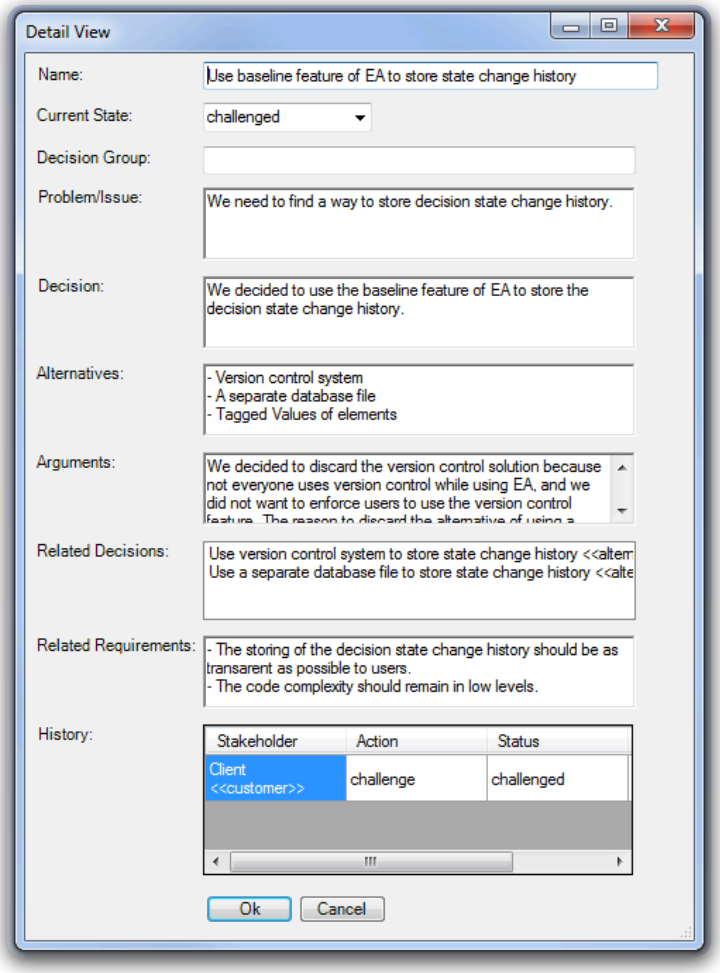


Figure 22: Detail view in Enterprise Architect.

3.2.2. Relationship

The Relationship viewpoint describes relationships between architecture decisions. Each decision is represented only once in this viewpoint. The Relationship view of this example is illustrated in Figure 23. It shows the current states of the decision and its related alternatives. It can be seen that the decision to use baselines was challenged and that a new idea emerged. The alternatives are connected with the *alternative for* relationship to the decision. The idea is implicitly an alternative to the final decision but it has not been evaluated yet. The Relationship view shows that there is currently an unsolved problem in the system or at least that some discussion is required.

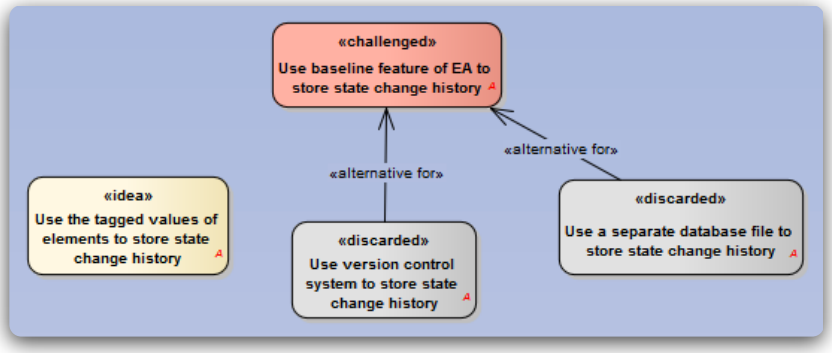


Figure 23: Relationship view in Enterprise Architect

3.2.3. Chronology

The Chronology viewpoint focusses on the evolution of decisions within a system. As part of Minor Milestone 2, the requirement to automatically generate a chronology view out of the stored decisions was implemented. However, in oder to implement this feature, the developer had to find a way detect and store state changes of decisions. While the detection was relatively easy to implement, the bigger problem was to find a suitable storage for the data.

As illustrated in Figure 24, three possible alternatives were identified, which are displayed as ideas in the diagram. The ideas were to use the baseline feature of EA, to use a version control system, or to use a separate database file. After the ideas have been evaluated, two alternatives were discarded and it was decided to use the baseline feature. After Major Milestone 1, the client had the opportunity to try-out the implemented feature. However, some usability issues were identified, which were caused by the decision to use baselines. Therefore, the decision was challenged by the client. In consequence, the developer came up with a new idea that has not been considered, which was to use tagged values.

3.2.4. Forces

As mentioned in the previous section, the initial ideas were evaluated. Therefore, a Forces view has been created, which allows to evaluate multiple alternatives for a decision against a set of concerns. The implementation of the Forces view is comprised of two parts. A configuration view and the actual table. The configuration view allows the user to easily create a Forces view via drag and drop and by relating concerns to and requirements (forces). The actual table is automatically updated each time the diagram

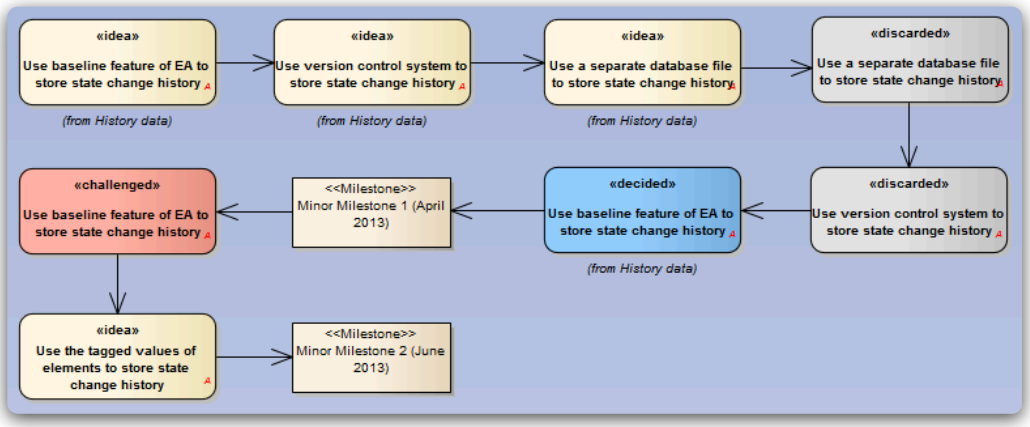


Figure 24: Chronology view in Enterprise Architect.

has been modified. The configuration view shown in Figure 25 contains the relevant alternatives, requirements and concerns for the decision. In this case, the relevant concerns are usability, code complexity, reliability, and performance. The requirements are that the solution should be easy to set-up and share without too much hassle for the users, it should be fast and reliable, and it should not increase the complexity of the code.

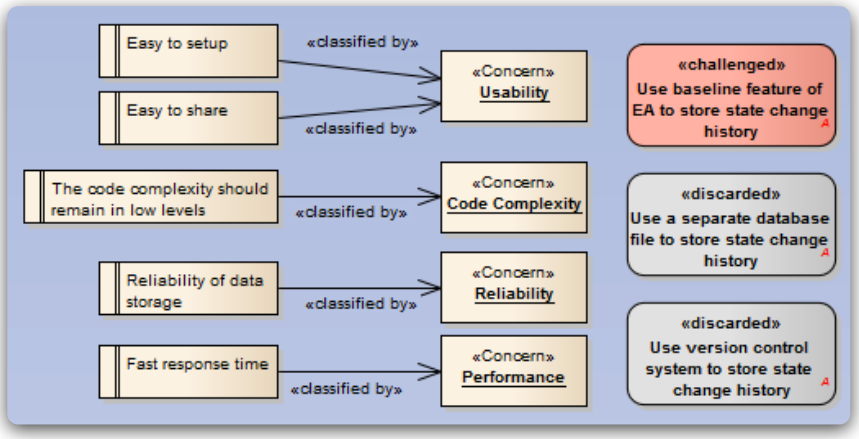
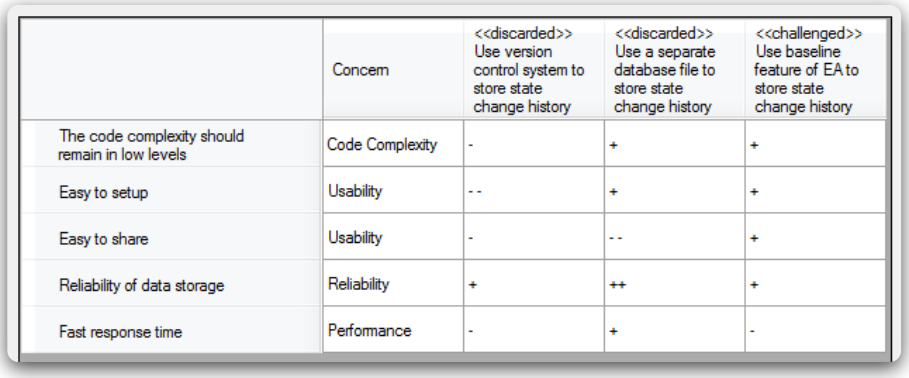


Figure 25: Configuration perspective on the Forces view.

Figure 26 depicts the actual Forces table. The left part of the table shows the considered requirements. This table has been used to evaluate the alternatives against the requirements. The rating schema is up to the user but we followed a standard `{++, +, ?, -, --}` schema. This schema is capable of indicating positive or negative impact of a decision alternative on a requirement, a question marks expresses uncertainty. It can be seen that using a database file has a positive impact on almost every requirement. However, it has a strong negative impact on usability as it requires additional effort to setup the database and makes it more difficult to integrate the add-in into an existing IT environment. On the other hand, the solution to use a version control system is negatively evaluated for almost every requirement. Setting up a version control system require a lot of effort and not all EA users are familiar with those systems. Furthermore, it is estimated that this alternative negatively affects code-complexity. he third alternative which is also the *final decision*, is rated with + for almost every requirement. The developer werde confident that this implementation is a good trade-off between usability and code complexity. Moreover, this solution does not require additional configuration as it is a inherent feature of Enterprise Architect. Nevertheless, reading and comparing baselines takes several seconds even for small Chronology viewpoints. However, the negative impact is limited as the Chronology view will not be generated frequently.



	Concern	<<discarded>> Use version control system to store state change history	<<discarded>> Use a separate database file to store state change history	<<challenged>> Use baseline feature of EA to store state change history
The code complexity should remain in low levels	Code Complexity	-	+	+
Easy to setup	Usability	- -	+	+
Easy to share	Usability	-	- -	+
Reliability of data storage	Reliability	+	++	+
Fast response time	Performance	-	+	-

Figure 26: The table part of the Forces view.

3.2.5. Stakeholder Involvement

Figure 27 shows the involvement of stakeholder in the decision-making process until Minor Milestone 2. As illustrated the developer proposed the three alternatives and the Scrum Master validated and discarded the alternatives. Figure 28 shows the involvement of stakeholders in the decision after Major Milestone 1. Compared to the previous diagram it can be seen that the client challenged the decision, while the developers proposed a new idea.

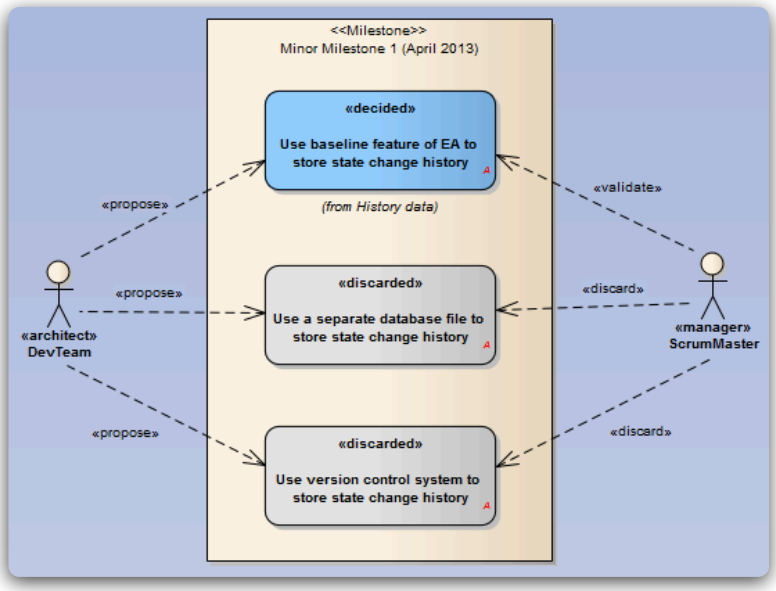


Figure 27: Stakeholders involvement before Minor Milestone 2.

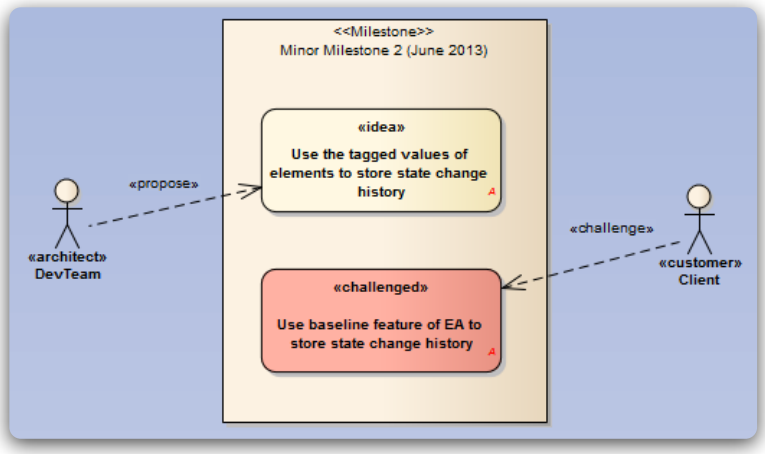


Figure 28: Stakeholders involvement after Major Milestone 1.

3.3. Additional Features

The add-in implements two additional features that increase its utility. The first one is traceability between architecture decisions and other architectural design elements, like components. The second feature is the ability to generate reports.

3.3.1. Tracing

The add-in provides a user-friendly way of tracing decisions to other elements, which has been also used for this example. After it has been decided to use the baseline feature of EA to store the state changes, the developers introduced a Baselines component in the architecture of the add-in. A trace can be created by adding the decisions in the same diagram as the component. After this a trace relationship between the two elements can be created by using the Trace-connector in the toolbox. Figure 29 shows the component view and the decision, which is traced to the Baseline component. Even if the decision is removed from the diagram, the decision is still traced to the component. However, we decided to leave the decision in the diagram and show/hide it via diagram filters. This allows us to get a good overview of the components and the underlying rationale.

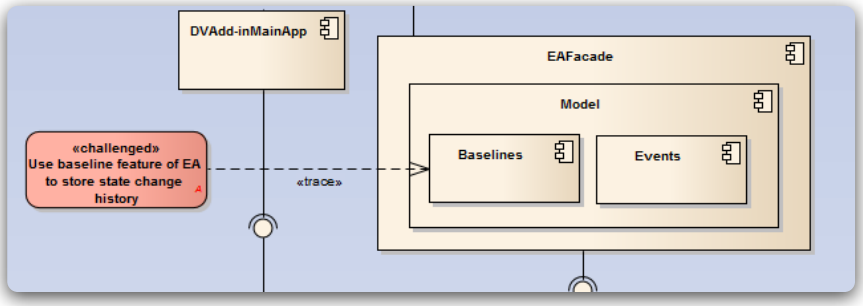


Figure 29: Trace between decision and baseline component.

The tracing is bi-directional, so the user can navigate from decisions to design elements and vice versa through the context menu of the decision element, as illustrated in Figure 30). When a trace is followed, the related diagram will open and the traced elements will be selected. In the case that an element has been added to multiple diagrams, the users is provided with a dialog to select the diagram he wants to see.

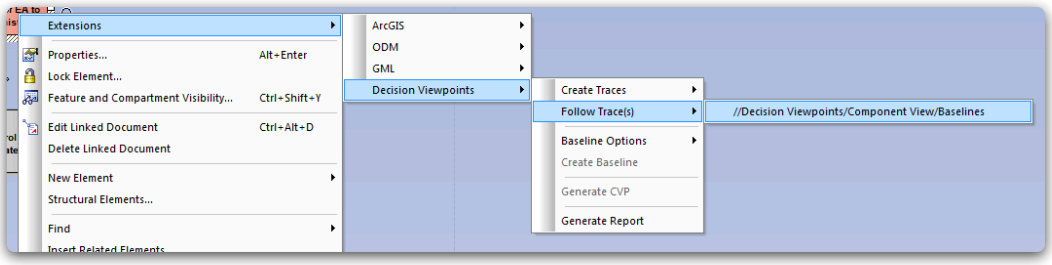


Figure 30: Following a trace through the context menu.

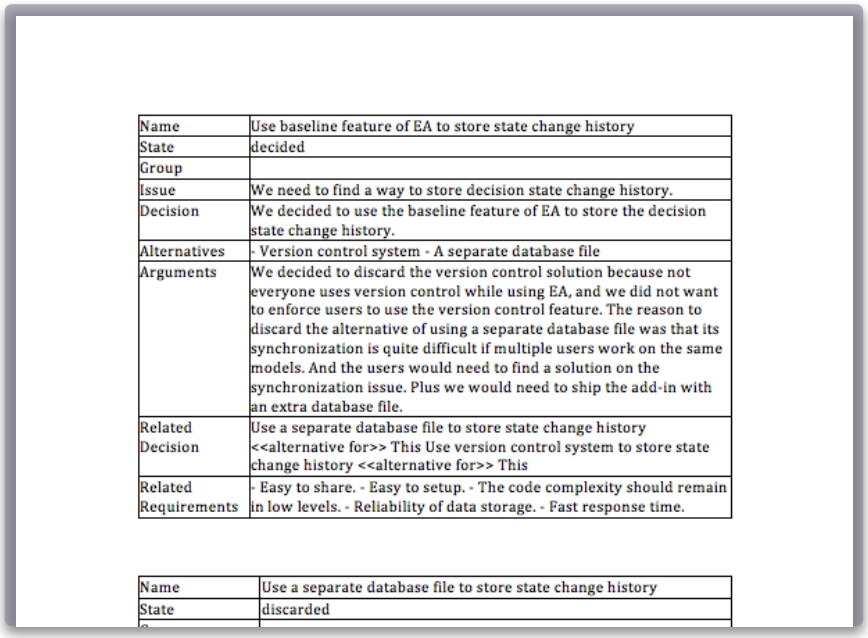
3.3.2. Model Validation

The decision documentation framework defines a set of constraints for each viewpoint. The purpose of the constraints is to enforce the semantic correctness and consistency. For example, decisions that have the state *idea* should not have any relationships to other decisions. This should emphasize the transient nature of ideas. As requested in the requirements, the add-in should provide the possibility to very if the decision model adheres to the constraints. As discussed in Section 2.5.4, it was specifically requested by the architects that the constraints are not strictly enforced during modeling but can be evoked on request. The add-in is integrated into the model validation functionality of Enterprise Architect.

The model validation function automatically verifies the open EA model against a set of rules. It is possible to extend the set of rules with custom rules via the add-in API. The model validation can be evoked through the menu of EA. At the end of the validation a list of warnings and errors is displayed. The example used in the section would not yield any warnings as it adheres to the rules defined in the framework.

3.3.3. Reporting

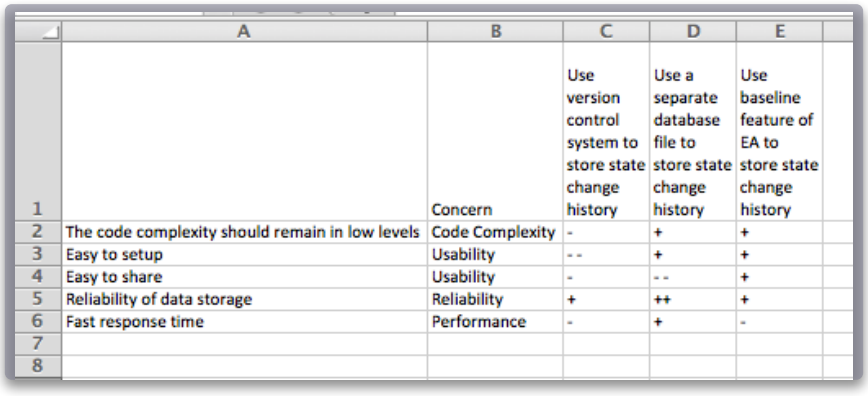
Three types of documents can be generated out of the decision documentation: Word, Excel, and PowerPoint. Each document contains specific type of information. Word and PowerPoint documents consists of the detail view for each decision, the views as diagrams, and the forces tables. Excel documents (cf. Figure 32) only contain Forces tables. Figure 31 and Figure 33 show a decision table in a generated Word and PowerPoint document . A decision table has exactly the same structure as the Detail viewpoint.



Name	Use baseline feature of EA to store state change history
State	decided
Group	
Issue	We need to find a way to store decision state change history.
Decision	We decided to use the baseline feature of EA to store the decision state change history.
Alternatives	- Version control system - A separate database file
Arguments	We decided to discard the version control solution because not everyone uses version control while using EA, and we did not want to enforce users to use the version control feature. The reason to discard the alternative of using a separate database file was that its synchronization is quite difficult if multiple users work on the same models. And the users would need to find a solution on the synchronization issue. Plus we would need to ship the add-in with an extra database file.
Related Decision	Use a separate database file to store state change history <<alternative for>> This Use version control system to store state change history <<alternative for>> This
Related Requirements	- Easy to share. - Easy to setup. - The code complexity should remain in low levels. - Reliability of data storage. - Fast response time.

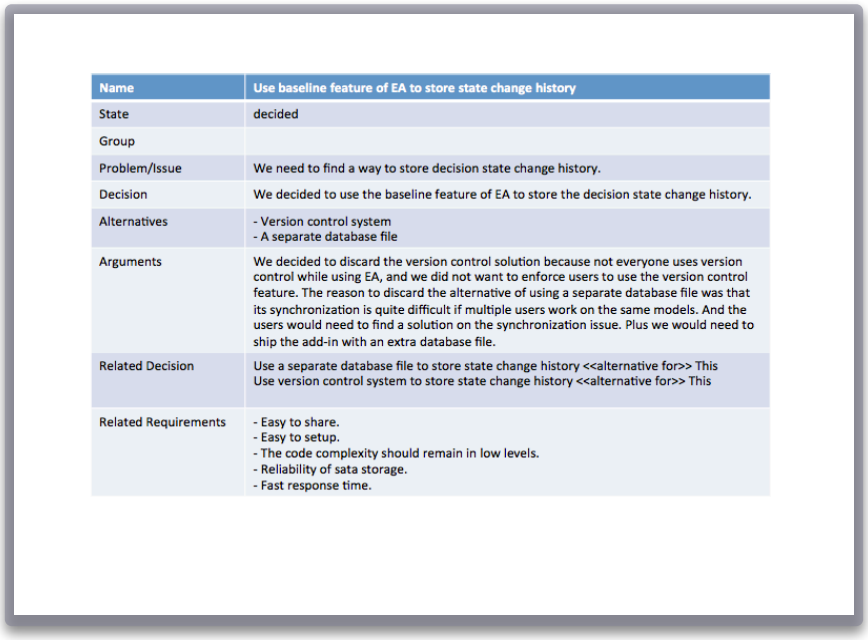
Name	Use a separate database file to store state change history
State	discarded

Figure 31: Generated Word document.



	A	B	C	D	E
1		Concern	Use version control system to store state change history	Use a separate database file to store state change history	Use baseline feature of EA to store state change history
2	The code complexity should remain in low levels	Code Complexity	-	+	+
3	Easy to setup	Usability	- -	+	+
4	Easy to share	Usability	-	- -	+
5	Reliability of data storage	Reliability	+	++	+
6	Fast response time	Performance	-	+	-
7					
8					

Figure 32: Generated Excel document.



Name	Use baseline feature of EA to store state change history
State	decided
Group	
Problem/Issue	We need to find a way to store decision state change history.
Decision	We decided to use the baseline feature of EA to store the decision state change history.
Alternatives	- Version control system - A separate database file
Arguments	We decided to discard the version control solution because not everyone uses version control while using EA, and we did not want to enforce users to use the version control feature. The reason to discard the alternative of using a separate database file was that its synchronization is quite difficult if multiple users work on the same models. And the users would need to find a solution on the synchronization issue. Plus we would need to ship the add-in with an extra database file.
Related Decision	Use a separate database file to store state change history <<alternative for>> This Use version control system to store state change history <<alternative for>> This
Related Requirements	- Easy to share. - Easy to setup. - The code complexity should remain in low levels. - Reliability of sata storage. - Fast response time.

Figure 33: Generated PowerPoint document.

4. Evaluation

We conducted an exploratory case study at our industry partner to assess the status quo of architecture decision documentation, to identify expectations of the ideal documentation approach, and to evaluate the advantages and limitations of the developed extension. According to Yin, case studies are preferable in situations, in which control of the variables is not possible or not required [30]. In our case the objective evaluation of usefulness and effectiveness of the decision viewpoint add-in in an isolated environment are not feasible. Project-related factors such as complexity, size, management requirements, or number of stakeholder influence not only the quantity and quality of documented decisions but also the willingness of the architect to use such a tool. Therefore it is important to understand the context in which the tool will be used and evaluated.

The focus of this case study is a preliminary investigation of the decision viewpoints add-in to seek new insights and to generate new ideas for our research. Because of this we chose to conduct an exploratory case study [31]. An exploratory study for tool evaluation suggests the use of qualitative evaluation approaches, which are targeted to establish tool appropriateness [32], i.e. how well a tool fits the needs and culture of an organization. Another aim of the research is to assess and describe the current state of decision documentation. Hence, the case study also has descriptive elements, as it is used to describe characteristics of a population or phenomenon being studied [31].

A fundamental step in case study research is the development of a research protocol. The protocol provides a description and justification of the procedures followed throughout the case study for collection and analysis of the raw data [31, 33]. A rigorous protocol increases the reliability of the case study research [30].

A protocol (i) can make the research concrete during the planning phase, (ii) it can be used as a basis to receive feedback in order to mitigate risks and threats of validity as early as possible, (iii) and act as a log where all conducted data collection and analysis is recorded [31]. Our protocol is outlined according to the case study protocol template proposed by Brereton et al. [34]. In order to design and conduct our case study we follow the guidelines of conducting a case study described by Runeson et al. [31]. As illustrated in Figure 34, they separate a case study into five major steps. Unlike other empirical methods, case studies is a very flexible research method and therefore there is a significant amount of iterations between the steps.

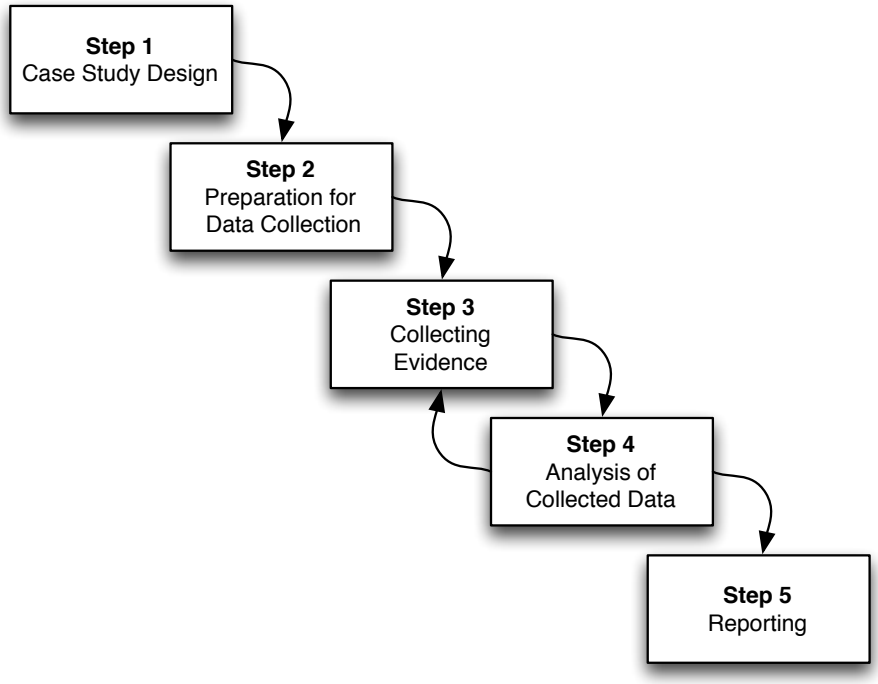


Figure 34: Case Study process as described by Runeson et al. [31].

Step 1 – Case study design: In the first step the reason for conducting a case study, the objectives and the research questions are defined. Furthermore, the study design is planned like selection of cases and choice of data collection instruments.

Step 2 – Preparation for data collection: In the second the case study protocol is written and the selected instruments are prepared, e.g. operationalizing research questions into interview questions or preparing observation forms. Furthermore, the instrument need to be validated in pilot studies.

Step 3 – Collecting Evidence: The defined protocol is executed and the data is being collected.

Step 4 – Analysis of collected data: The data analysis procedures are applied to the data. If insufficient data was collected it is possible to execute step 3 again collect additional data. In some cases it might be even possible to go back to the second step to define additional methods of data collection.

Step 5 – Reporting: The study and its conclusion are reported and disseminated in appropriate format.

4.1. Research Questions

The main research goal of our study is to...

> ...analyze *the decision viewpoint technology* for the purpose of *evaluation* with respect to *advantages and limitations* from the point of view of *software architects* in the context of the *ABB* company environment.

We want to understand the factors that would support or prevent the adoption of a more systematic approach of ADD, in this case by using the decision viewpoint technology. An important aspect of this research is to explore to what extent the proposed add-in can be used by architects to document decisions in an efficient and user-friendly way. In order to explore these advantages and limitations of the decision viewpoint technology, we also need to investigate current ADD approaches used in the selected Business Units (BU) of ABB and to get an understanding of what ABB architects perceive as an ideal solution for tool-supported ADD.

Based on our study goals (i.e. evaluate the decision viewpoint technology), we derive three research questions.

RQ1 What is the status-quo of architectural decision documentation in the selected business units of ABB?

RQ2 What is the ideal AD documentation approach from the perspective of ABB architects?

RQ3 How do ABB architects perceive the advantages and limitations of the decision viewpoint technology?

RQ1 provides us with information about the current practices for decision documentation in ABB. RQ2 helps us to identify the ideal AD documentation approach for ABB architects. We ask RQ3 to identify the advantages and limitations of the decision viewpoints technology and discover new insights about tool-support for ADD.

4.2. Case and Units of Analysis

We conducted a single case, embedded case study. In embedded case-studies "multiple units of analysis are studied within a case" and are an appropriate design for explorative case studies when there is no established theoretical framework[31]. Our concrete case is the documentation of software architecture decisions by four architects of different business units of a large industrial-automation company. As each architect documents decisions for a different project,each documentation activity is considered to be a unit of analysis in itself. The object of our study is the Decision Viewpoints add-in, which is a software project that extends the capabilities of EA to facilitate the documentation of ADs.

The scope of our study is limited to the Business Units (BUs) of ABB to which the participating architects belong. We will collect evidence by observing and interviewing software architects and by looking at real Architecture Decision (AD) documentation. The participants are perceived as experts in software architecture with substantial practical experience.

4.3. Design

To strengthen our research findings, we used multiple sources of data [33]. These include existing AD documentation in ABB as an independent data collection technique, semi-structured interviews with ABB architects during requirements elicitation and during the field study, and direct observation of ABB architects throughout the tool session. The selected sources of data belong to the ones most commonly used in case studies [30]. We explored our research questions by using the following methods and sources of data. Table 1 shows the mapping between selected sources of data and defined research questions.

Table 1: Mapping between sources of data and RQs.

	Requirement elicitation interviews	Existing decision document	Pre-tool session interview	Observations from tool session	Post-tool session interview
RQ1 Status Quo of decision documentation	×	×	×		
RQ2 Ideal ADD Approach	×		×	×	
RQ3 Advantages and Limitations of the Add-in				×	×

Requirement elicitation interviews:

Eliciting requirements for the tool from software architects, helped us to understand both the status quo of AD documentation and the ideal tool. We conducted individual exploratory interviews with ABB architects. An exploratory interview is well-suited to elicit new and innovative requirements[35] and therefore can be used to get insights about the ideal ADD approach.

Existing decision documentation:
Analyzing existing decision documentation helped us to understand the status quo of AD documentation at ABB. Moreover, it enabled us to collect details to corroborate information collected from other sources.

Tool sessions:
The purpose of tool-sessions was for architects to familiarize with the framework and tool, and for us to observe how the architects used the tool. Participant observation refers to a research that involves social interaction between the researcher and informants in the milieu of the latter, during which data are systematically and unobtrusively collected [36]. According to Yin [30], observational evidence is often more useful in providing additional information about the topic being studied. Analyzing the results of the observations helped us to identify advantages and limitations of the decision viewpoint technology and thus contributed to RQ3.

Pre- and post-tool session interviews:
We conducted interviews with ABB architects before and after they have used the tool to document architectural decisions from a concrete project. Semi-structured interviews before and after the tool-sessions provided us with insights into human affairs or behavioral events [30] and helped us to better understand the context in which the add-in is evaluated.

4.4. Data Collection

The data collection was conducted in three phases.

1. Requirements elicitation interviews were conducted at the beginning of this project.
2. Collecting existing documentation before and after the on-site study.
3. During the on-site study we conducted the tool-session and in-depth interviews.

4.4.1. Requirement Elicitation Interview

Due to the geographically dispersion of participants and researchers, the requirements elicitation interviews were conducted via telephone. We chose a semi-structured interview theme in order to elicit new or innovative requirements. We recorded the interview if the participant had agreed. If this was not the case, one of the researcher took extensive notes. The interview guide can be found in Section B.1. In total, we collected data from four architects. We conducted telephone interviews with three of them. One architect filled out the interview guide offline.

4.4.2. Existing decision documentation

To understand the status quo of AD documentation at ABB, we asked for existing decision documentation. This also proved helpful to direct questions of the in-depth interviews. The documentation that we received from ABB contained architectural design decisions for two systems and are based on real products of ABB. However, the decisions have been obfuscated to hide any product-specific information or component names. These decisions are documented in natural language. In the analysis of these documents, we looked for what kind of information has been documented (e.g. problem, solution, how many alternatives were considered, relationships, stakeholders, etc.). We also received modeling guidelines, templates that were being used to document decisions in concrete projects and an Excel forms for evaluating alternatives against concerns.

4.4.3. On-site Study

Most of the data collection took place during a company visit in Ladenburg (Germany) on the 8th and 9th of July 2013. Three researchers from the RUG team conducted the study in the ABB corporate research center. In total, we had individual sessions with five architects. All sessions followed the same schema as illustrated in Figure 35. We first interviewed the participant about the status-quo of decision documentation. The first interview was followed by a brief presentation about the Decision Documentation Framework and the main features of the add-in, including those features that are not described in the viewpoints (e.g. typical project structure, traceability, automatic generation of views). Three of the participants used the tool to document decisions of a recent software project. In the end of each session, we conducted another interview about their experience and their opinion about the advantages and limitations of the add-in. Each session was completely recorded and transcribed. The interview guide for the first interview can be found in Section B.2; the guide for the second interview in Section B.3.

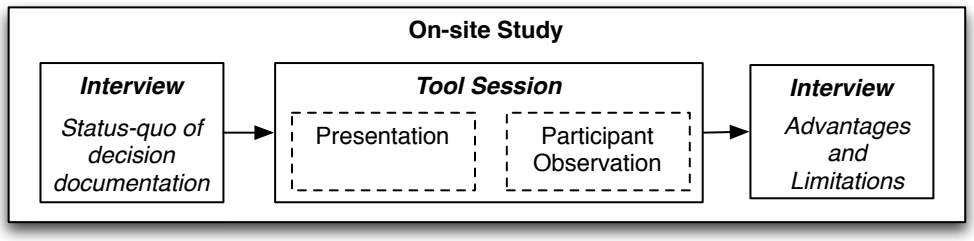


Figure 35: Schema of an individual session.

4.4.4. Subjects and Projects

All five architects can be classified as experienced software architects being actively involved in architecture activities at the time the study was conducted. The participants had between 3 and 18 years of experience in the field of software architecture (average 11 years) and 10 to 24 years of experience in software development. One of them was a certified TOGAF[12] senior architect. The participants mentioned to have experience in the the following domains, Information Systems, Enterprise Applications, Corporate Research, Control Systems, Process Automation, Power Generations and Industrial Software Architectures.

Architect 1: The first participant primarily reported about a corporate consulting project. He was the lead architect and project manager. The project had many stakeholders from different units of the organization. Documentation was a main deliverable of the project. During the one year project, 20-30 decisions were explicitly documented.

Architect 2: The second architect reported about a long-term project in the domain of process automation with external clients. He was one of many software architects working on this project, being involved in architecture decisions on a daily basis but he reported that only the "bigger" decisions, which occur every 1-2 weeks, are worth documenting. He mentioned that they documented most of the decisions in the beginning of the project but due to time constraints were not able to document that many decisions recently.

Architect 3: The third architect reported about an industrial control system, in which he had the role of a system architect. The project combined software and hardware. He was involved in 5-6 major decisions within a timespan of three years.

Architect 4: The fourth architect reported about a large corporate research, in which he is involved as lead architect. In the project they were explicitly documenting decisions. In the first six month 15 decisions were partially documented. Some decisions only identified topics that would need a decision in a later phase of the project and thus served the purpose of placeholders.

Architect 5: The fifth architect was working in the field of industrial software architectures as an architect and project leader. He reported about two projects with a duration of one year. In the first project 20-30 decisions were documented; in the second project 30-40 decisions.

[12]The Open Group's Architecture Forum

4.5. Analysis

Since we collected qualitative data, we chose to apply grounded theory to analyze the data. In recent years, grounded theory has gained more attention in software engineering related research [36, 37]. Grounded theory is a labor intensive analysis method but "is inherently explorative in nature, as it promotes the analysis of data without predetermined ideas about potential findings [36]" and thus suitable for this study. In grounded theory, qualitative data is systematically analyzed to generate findings that are supported in multiple ways by the data[37]. The classic theory generation method in the qualitative literature is the constant comparison method[36], in which codes are attached to pieces of text that are relevant to a particular topic or idea. Categories and concepts emerge slowly by constantly comparing coded text found in the data to previously assigned codes.

4.5.1. Analysis Procedure

We used constant comparison for analysis, please refer to Adolph et al.[37] for an in depth explanation of the terms code, concept, and category. The steps we followed during the data analysis are briefly described. Note that steps two to four are performed iteratively.

1. **Transcription and Format:** The data was exclusively stored digitally in an online shared folder. The recorded interviews were transcribed by two researchers that participated in the interview. The transcribed interviews were converted into spreadsheets to ensure a uniform coding of the data. We used an online office platform for collaboration and to support simultaneous analysis.
2. **Coding data:** All interviews and the collected documentation were intensively studied. We avoided the use of pre-formed codes as the objectives of the analysis were not clear ahead of time. Potential codes were announced, explained and discussed in a shared document during analysis. Responses of the participant that were related to one of the research questions were labeled in a separate column in the spreadsheet. Relevant text passages in the existing documented were labeled with PDF memos and text highlighting.
3. **Identify concepts:** During the coding and constant comparison the researcher gets a better understanding of the data he is looking at. The identified codes were grouped into related higher level concepts. We used a shared spreadsheet to compare the collected codes and the related text passages and combined those that had commonalities. In our case, a concept is a representation of a pattern of behavior, a recurring statement or an expression of an opinion.
4. **Classify concepts into categories:** After repeating step two and three multiple times, a set of concepts emerged. The concepts and the related data were intensively studied and discussed. Related concepts formed a category. A finding that is supported by a concept that can be traced to more than one interview is more reliable.

4.5.2. Categories

Table 2 maps the categories, identified in step four of the analysis procedure, to the sources of evidence, in which they were observed. If the category is mapped to an architect the evidence was observed during the company-visit. The column requirements means that the evidence is part of the requirement specification or was observed during the requirement elicitation interviews. The column existing documentation refers to documents that we have received from ABB.

Table 2: Mapping of categories to sources of evidence.

		Architect 1†	**Architect 2**	**Architect 3**	**Architect 4**	**Architect 5**†	**Requirements**	**Existing Documentation**	**Research Question**
(**cat**^{1})	Familiar with architecture decisions	×	×	×	×	×	×		RQ1
(**cat**^{2})	Primarily documented "'major" decisions	×	×	×					RQ1
(**cat**^{3})	Statement about approach	×	×	×	×				RQ1
(**cat**^{4})	Intensive-phase of preparation	×	×			×			RQ1
(**cat**^{5})	Quantitative evaluation of alternatives		×	×	×	×	×		RQ1
(**cat**^{6})	Time required to document decisions	×	×		×	×			RQ1
(**cat**^{7})	Tools and templates	×	×		×	×	×		RQ1
(**cat**^{8})	Maintenance of documentation is hard	×	×						RQ1
(**cat**^{9})	Fields and information captured	×	×				×		RQ1
(**cat**^{10})	No direct perceived benefit of ADD	×		×	×	×			RQ1
(**cat**^{11})	Decisions are documented for communication	×	×	×	×	×			RQ1
(**cat**^{12})	Lean and flexible documentation	×	×	×	×	×			RQ2
(**cat**^{13})	Tool should not patronize user	×							RQ2
(**cat**^{14})	Limited time for documentation	×	×	×	×		×		RQ2
(**cat**^{15})	Compatibility with project-specific guidelines		×	×	×				RQ2
(**cat**^{16})	Integration into Platform	×	×	×	×	×	×		RQ2
(**cat**^{17})	Sharing of technology evaluations		×	×	×				RQ2

(**cat**^{18}) Stakeholder specific reports		×				×	×	RQ2
(**cat**^{19}) Time savings		×			×			RQ3
(**cat**^{20}) Potential early adopter	×			×	×			RQ3
(**cat**^{21}) Different viewpoints to separate concerns	×							RQ3
(**cat**^{22}) Forces VP is useful for technology evaluation		×	×	×	×			RQ3
(**cat**^{23}) Forces VP does not provide enough possibilities to capture rationale	×							RQ3
(**cat**^{24}) Limited value of SIVP		×	×	×	×			RQ3
(**cat**^{25}) Reluctance towards accountability introduced by SIVP	×	×	×					RQ3
(**cat**^{26}) Approved CVP	×	×		×				RQ3
(**cat**^{27}) Too much effort to create CVP					×			RQ3
(**cat**^{28}) Auto-generation is problematic for maintenance		×	×					RQ3
(**cat**^{29}) Ambiguous scope of chronology viewpoint	×		×					RQ3
(**cat**^{30}) Relationship VP provides good overview		×	×	×	×			RQ3
(**cat**^{31}) Semantic of state, relations and traces unclear	×	×	×		×			RQ3
(**cat**^{32}) Detail VP is very useful	×		×	×	×			RQ3
(**cat**^{33}) Integration into EA	×	×	×	×	×			RQ3
(**cat**^{34}) Approved ability to trace decisions to modeling elements	×	×	×	×	×			RQ3
(**cat**^{35}) Difficulty to articulate decision topics, alternatives and outcome	×	×	×	×	×			RQ3

† Did not participate in requirement elicitation.

4.5.3. RQ1 - Status-Quo of Decision Documentation

We are describing the status quo of architectural decision documentation in recent software projects in various Business Units of ABB.

As indicated by Table 2 all architects were familiar with the concept of AD and documented decisions (**cat**^{1}). In fact, it was mentioned during the interviews that the

awareness for ADD is increasing since the past 2-3 years. The number of decisions that the architects were involved in did not depend on the duration of the project. They reported up to six decisions for a three year project or 40 decisions for a one year project (cf. Section 4.4.4).

The architects reported that they are mostly concerned about the *"major"* decisions of the systems and that these where the ones they have documented (**cat^{2}**). However, there was no common description or characteristics that would classify these *"major"* decisions. Often they were just referred to as *"high-level"*, *"fundamental"* or *"big"* decisions. One mentioned that these are the decisions *"that are causing discussion between stakeholders and require common agreement"*.

Although there was no company-wide approach for architecture decisions, the overall approach was similar (**cat^{3}**). Usually, the decision making process started with the identification of the problem or decision topic followed by preparation phase, in which they performed requirement analysis, stakeholder interviews, design space exploration (**cat^{4}**). Four architects used a quantitative matrix-approach to evaluate alternatives against concerns (**cat^{5}**), one preferred a qualitative approach using scenarios. After the preparation phase the decisions were made or approved during meetings with stakeholders. It was reported that the preparation phase for those *"major"* decisions is very time-consuming. Timespans from 1 week to 8 month were reported. The documentation of the decision itself was comparably short (**cat^{6}**); in general nor more than a day. One participant reported that it takes him roughly one week to document a decision but this included multiple iterations and reviews. An explanation for the short documentation phase is that the information is already present as it was gathered and produced during the preparation phase.

There was no common format in which decisions where documented or which tools were used for documentation (**cat^{7}**). Decisions were documented in PowerPoint, Excel, Word, Enterprise Architect. Most often decisions were documented by using a template, those templates were adapted to the specific needs of the project with inspiration from Zimmermann's y-approach or the ISO/IEC 42010 standard(**cat^{9}**). The decision documentation that we received was documented using the y-approach[13], which is a minimalistic decision template. For most decisions the forces, alternatives and argumentation were documented. Another template we received, specifically tailored for technology evaluations, also captured stakeholders, risks and a revision history. Furthermore, it was also mentioned that decisions are documented as a loose collection of documents in a documented management system in combination with one summary slide. The summary presented the outcome of the decision and listed the pros and cons. Since such tools are rather ineffective to manage decisions, maintaing decisions and keeping track of their progress was a problem (**cat^{8}**). This problem became evident in situations when related requirements changed or new information about a technology became available. This shows a clear need for proper tool-support, which is illustrated by the following quote:

[13] http://www.sei.cmu.edu/library/assets/presentations/zimmermann-saturn2012.pdf [Online accessed: 25/08/13]

"Yes, in the absence of good support in tools like Enterprise Architect we fell back to Powerpoint tables.".

From the interviews we have seen that the documented decisions are primarily used for communication(**cat^{11}**). For example decisions are shown in stakeholder-meetings for approval. In some cases decisions were also used to inform developers but it was mentioned that they are usually interested in the outcome of a decision rather than the underlying rationale. We found that in projects without these explicit stakeholder meetings the directs benefits of decision documentation were perceived lower(**cat^{10}**). The advantages of having explicit decisions were seen during the evolution of the system, e.g. knowledge vaporization and system understanding. It was reported that due to time-pressure decisions were not documented in these projects. In contract, in those projects where the client was actively participating in review meetings, the architect put a lot of emphasize on decision documentation and it was considered a positive return on investment. *"The feedback was very positive much more appreciated than pure component and connector documentation. It was perceived to the point and I talked about derived decisions and documented them in a consumable way. That made the reviewing quite efficient."*.

4.5.4. RQ2 - Ideal AD Documentation Approach

To answer RQ2, we looked at the elicited requirements (cf. Section 2.3), the pre- and post-tool session interviews and observations during the tool-sessions. We do not aim to report a set of features as they have been reported by Jansen et. al [22] and are explicitly described in the requirement specification of this project.

All architects reported that a lean and flexible documentation approach is fundamental (**cat^{12}**). The tool should not enforce documentation but should offer the flexibility to document only certain aspects of decision. This means that the architect can choose how much documentation is really necessary and which information is critical to understand the rationale of the decision (**cat^{13}**). This is supported by the finding that the participants only documented those decisions that they considered to be important (**cat^{2}**). Four of the architects mentioned during their interview that hard time constraints limited the amount of decisions that were documented, therefore documentation needs to be efficient as possible (**cat^{14}**). One architect summarized this as follows: *"Do not give them too many options and too many things to do. Try to make them efficient in their work by minimizing the time they have to spend in creating a decent decision log"*. Frameworks and models with complex semantics, rules and formalisms are not seen beneficial as the user would have to spend a lot of effort to be correct and precise.The overall approach should be flexible enough in order to be integrated with corporate guidelines and existing development processes (**cat^{15}**).

According to the participants, the decision documentation tool should be integrated into existing platforms (**cat^{16}**), such as a modeling tool. They want to be able to relate decisions to their existing design artifacts and the captured requirements. This would

reduce maintenance effort as changes in the models and requirements are synchronized to the decisions and sanity checks can be performed. Overall, automatic generation of documentation was requested in the requirements interviews (req. **A2, A5**) and the possibility to relate decisions to external sources, such as document management systems. The promise is that architectural knowledge documentation can be speed up by aggregating information, e.g. technology evaluations, rather than duplicating work. Additionally, there is a need to effectively share architectural knowledge across business units, in particularly the results of technology evaluations (**cat**$^{\mathbf{17}}$). A lot of time is spent on evaluating technologies against certain criteria. A shared repository of these evaluations could provide valuable insights on how other architects have assessed a technology and which risks they have foreseen.

As indicated by (**cat**$^{\mathbf{17}}$), the tool needs the ability to generate stakeholder-specific reports. This correlates with the fact that the main purpose of decision documentation within the observed organization was communication with stakeholders (**cat**$^{\mathbf{10}}$). It was mentioned that a lot of effort is spent to optimize the reports for non-technical stakeholders, that do not require the level of technical detail that a developer would expect. A comment of an architect was *"Stakeholder are really on the other side, you have marketing people and management there and you really need to adapt to their mind"*. On the other hand, the developer might not be interested in financial aspects of a technology decision.

4.5.5. RQ3 - Advantages and Limitations of the Add-in

We received very positive feedback about the current state of the add-in despite the fact that not all features were implemented. The tool that was being assessed only contained a draft implementation of the Forces viewpoint. The Detail viewpoint and the ability to generate reports were not implemented yet. Nevertheless, three of the architects stated that the presented tool would allow them to document decisions faster compared to their current approach (**cat**$^{\mathbf{19}}$). Three participants also mentioned that they would immediately start using a stable and complete version of this add-in in practice (**cat**$^{\mathbf{20}}$).

The approach to separate concerns by having different viewpoints was widely accepted (**cat**$^{\mathbf{21}}$), which is also supported by the fact that there was no common agreement on the usefulness of particular viewpoints. Some viewpoints were identified as the most important features, while others did not see an immediate benefit in practice for the same viewpoint. The fact that there is no agreement on the usefulness of certain viewpoints but that each architect considered at least one of them as useful supports the concept of views and viewpoints to address different stakeholder-concerns. This finding perfectly compatible with the decision framework and the tooling as it is up to the architect which viewpoints to use.

The Forces viewpoint was seen as very useful as it supports them in their main activities during decision-making, which is technology evaluation (**cat**$^{\mathbf{22}}$) using a concern-alternative matrix. However, we received one comment regarding the limited capabilities to manipulate the table. This included the possibility to assign colors, to group, to filter,

to expand and to collapse forces and alternatives. The table does not provide enough possibilities to analyze the data especially if many alternatives and forces are considered. The Forces viewpoint was not appreciated by one architect as he evaluates alternatives using qualitative methods, such as scenarios (**cat**^{23}).

The comments about the Stakeholder Involvement viewpoint were mixed. Four architects reported that they would probably not use this viewpoint in their current project as they do not see an immediate value(**cat**^{25}). On the other hand, one architect saw the advantage *"of having a connection from project management to architecture decision-making. This is the benefit of having this viewpoint"*. However, three participants had doubts that the accountability of stakeholders introduced by this viewpoint (**cat**^{25}) would create tension in the project.

The implementation of the Chronological viewpoint was considered positively (**cat**^{28}). However, they stated that the effort to create this viewpoint manually would not justify the insights gained from this viewpoint (**cat**^{27}). Although the viewpoint comes with an auto-generation functionality, the participants saw some problems with the maintenance of the documentation (**cat**^{28}) as there is a risk that the auto-generation overwrites manual changes. Furthermore, it was criticized that the scope of the viewpoint can be improved as it combines the overall evolution of all decisions with the history of individual decisions (**cat**^{29}). The viewpoint has the risks to contain too many elements and thus becomes too large.

The Relationship viewpoint was considered very positive (**cat**^{30}) as it provides a good overview of the decisions in the project. As indicated by (**cat**^{31}) the concept of decision states and the meaning of relations are not intuitively understood by the architects and require some training.

The ability to relate decisions with other modeling elements was approved by all architects; *"That's a dream coming true"* (**cat**^{34}). The Detail viewpoint was only shown on paper as it was not implemented yet. Still the comments were very positive (**cat**^{32}), since the architects were used to document their decisions in template-form.

The tight integration with EA was praised and that the add-in feels like a natural part of it (**cat**^{33}). However, a general concern that was raised by all participants was that the add-in does not clearly distinguishes between the concept of a decision topic and the outcome of a decision (**cat**^{35}). This makes it difficult to explicitly model decision points and articulate alternatives.

4.6. Limitations

The validity of a study is an indicator of the trustworthiness of the results and to what extent the results are true and not biased by the researchers' subjective point of view [31]. We follow the classic scheme proposed by Yin et al. that distinguishes between four aspects of validity [30]. *Construct validity* reflects to what extent the operational measures that are studied really represent what was intended to be studied. *Internal*

validity reflects the extent to which a causal conclusion based on a study is actually present and not affected by an unconsidered factors. *External validity* reflects the extent of generalizability of the findings, and to what extent the findings are of interest to other people outside the investigated case. *Reliability* is concerned to what extent the operations of a study can be repeated , with the same results and thus to what extent the study is dependent on the specific researchers.

4.6.1. Construct Validity

To ensure that our operational measures are suitable, we established a research protocol which was reviewed by two researchers with experience in conducting case-study research. We conducted two pilot-studies to train the interviewers and to identify any problems with the questions [31]. The subject of the pilot study were both experienced software developers. We used multiple of sources of data in order to limit the effects of one interpretation of a single data source [31], this included interviews, participant-observations and the analysis of work artifacts. We also took different perspectives into account and investigated differences between projects and business units. This improves the strength of our conclusions. In our study design we ensured that the interview about the status quo is not biased by the demonstration of the add-in. Since we were conducting an explorative study, we did not use a pre-formed set of codes to leave the outcome as open as possible.

4.6.2. External Validity

Because statistically representative samples can typically not be achieved in cases studies, the emphasis is usually put on analytical generalization, thus an explanation why the findings are representative for other cases with common characteristics [31]. With respect to our findings for RQ1 and RQ2, we believe that our findings are generalizable to a larger population as they are supported by other studies that were conducted in different domains and with different subject [18, 38]. The scope of our study is limited to the selected Business Units of ABB. The influence of the company culture is evident due to the fact that the company was raising awareness for architecture decision documentation. However, we believe that the subject population in the study is representative for the larger population of experienced software architects. The participants of the study were experience software architect with multiple-years of experience in architecting software systems. There is a risk that the cultural background influenced the study results as all participants were working in Germany. However, the company is multinational and there is a lot of information exchange with different countries or continents. With respect to RQ3, external validity is limited as further replications with multiple-cases are necessary [30].

4.6.3. Internal Validity

Since we are conducting an exploratory case-study we are not aiming to establish any causal relationships in our data, therefore the concept of internal validity is not a risk for this study [30].

4.6.4. Reliability

We addressed the reliability of our study by defining a rigor study protocol and interview guides [31]. These can be used to replicate the data collection procedure. In addition, we created a case-study database that contains all collected data both raw interview recordings and transcriptions, so that the data analysis can be verified and traced. A potential risk is that we biased our findings towards a positive outcome. We mitigated this risk by asking open-ended questions like "To what extent would you use the add-in during architecture decision documentation?". As follow-up questions, the interviewer asked the participant to motivate their answers, or to go more into detail. To mitigate the risk of suggestive questions and to make sure that all important topics would be covered, we prepared an interview guide [31]. However, the risk of a researcher bias also affects the analysis. The data coding was conducted by two researchers to increase the reliability of data interpretation. The findings and their origin in the data were discussed with a third researcher that did not participate in the coding procedure to get an unbiased perspective.

5. Discussion

In this section, the findings are discussed and compared to results of existing approaches in the literature. Furthermore, the developed tool is closely examined and compared to the initial vision, use cases and key drivers.

5.1. Lack of Tool-Support and Standard Approach

The analysis of RQ1 showed that architecture decisions are documented in practice, which indicates that the importance of architecture decisions in the architecting process is receiving raising awareness also among practitioners. The finding is supported by a survey conducted by Tang et al. in 2005 that showed that decisions are indeed documented, contrary to popular believe [38]. However, this study also showed that the justification of a decision and the advantages over an alternative are usually not documented. However, our findings showed that the documented architectural knowledge varied between projects. Some architects documented decisions thoroughly with alternatives, justification and consequences, while others mostly focussed solely on the outcome.

An explanation for that could be a missing standard approach and the use of general purpose tools to document decisions. Unlike general purpose tools, a specialized tool can present the architect with a template and ask him to provide more information about a decision thus increasing the quality and amount of captured knowledge. Furthermore, non-specialized tools are rather ineffective for creating and maintaining architecture decisions. A finding that is supported also by a recent study by Anvaari et al. on decision making in the norwegian electricity industry. In their case, the architects also struggled to keep the documentation up to date [39]. Overall, the lack of good tool support might be discouraging for architects to capture knowledge in the first place.

5.2. Sharing and Reusing AK

RQ2 and RQ1 show that the ability to share AK with stakeholders or non-tool users is an fundamental use-case for architects. It is important that there is a possibility to extract architectural knowledge from the captured decisions and to generate stakeholder-specific reports. This becomes evident when considered that communicating decisions was the primary motivation for documentation. This finding is also supported by Farenhorst et al. who list stakeholder-specific content as one of the characteristics for architectural knowledge sharing tools.

We have seen in RQ1 that a lot of time is spent on the analysis of decisions, e.g. identifying architecturally significant requirements, find candidate solutions and evaluate those solutions. This is aligned with the findings from van Heesch and Avgeriou on the reasoning process of professional architects [40]. Despite the effort that is spent on preparing a decision, we found that decisions are not shared within the company but are only consumed within the context of a project. However, there is a potential to utilize ar-

chitecture decisions across projects. Especially technology evaluations could provide a good basis for upcoming decisions as the effort for collecting and researching information about candidate solutions could be significantly reduced. Of course, those evaluations cannot be applied without a critical reflection as the context and requirements of projects are not the same. What works in one case may not work in another but evaluations can be used as a source of inspiration (which alternatives were considered) and information (how where they evaluated). In our tools these evaluations are embedded in the Forces viewpoint but besides a reporting functionality there is no dedicated support to share and reuse the views across projects. A question for future research could be, if it is possible to make the reasoning process more efficient by sharing and reusing AK and in particular Forces views across projects.

5.3. Efficiency, Flexibility and Perceived Benefits

RQ2 showed that there is only limited time available to be spent on documentation efforts. This is supported by the study conducted by Tang et al., who identified time-pressure as one of the main barriers to documenting decisions [38]. However, time-pressure can be seen as an indicator that the architects did not see enough direct benefits to make decision documentation a higher priority task. Although, all participants reported long-term benefits of decision documentation such as avoiding knowledge vaporization, the immediate benefits of documenting decisions were not that evident. Architects only mentioned direct benefits in those projects where decisions were immediately used for stakeholder reviews, or where the project focussed on extending the knowledge-base of the company rather than developing a new project, e.g. by creating architecture prototypes of emerging technologies.

This is aligned with our finding that the most requested characteristics of the ideal approach are flexibility and efficiency. The effort of documenting decisions should be as low as possible, for instance by generating parts of the documentation. Generating viewpoints would be a possibility to increase the quality of documentation (i.e. completeness) without requiring additional effort. This could make the decision documentation more suitable for a wider audience as it is not always predictable which stakeholder will eventually consume the documentation and for which information he will be looking.

Furthermore, the approach should be flexible enough to be integrated with existing software processes, modeling guidelines and tools that are used in the company. Another dimension of flexibility, is that the tool should let the architect decide when to document what. Architects may choose to only document the major decisions of the system and omit the smaller ones. They may also chose to do this immediately or in retrospective after a milestone has been reached and the decisions are more stable. In general, the tool should not patronize its users by enforcing a certain way of documentation. This was also reported by Farenhorst et al., they state that due to the creative nature of architecting, tools should follow a descriptive approach nature rather than being prescriptive [18]. From a tool development perspective, a question for future research would be how to

further minimize the effort of documentation, for instance by automating or generating parts of the documentation.

5.4. Reinforced Findings about the Framework

One of the motivations behind the Decision Viewpoint Framework is to be able to address different concerns and to support all architecture life-cycle phases, so that the architect may choose not to document all views, but choose a subset depending on the characteristics of the concrete project and the available time [5]. The analysis of RQ3 showed that this goal has been accomplished as the participants considered different viewpoints to be useful for their project. This emphasizes the importance of providing different perspectives on AK as it is impossible to accommodate every concern with a single view on AD.

Our study confirms multiple findings about the framework identified by the evaluation of the decision documentation framework. This includes that the stakeholders concordantly stated that the relationship view is well suited to get an overview of the decisions made [5]. We could also confirm that the effort for creating the chronological view does not pay off in every project, e.g. if many stakeholders are involved or if projects are subject to long-term evolution [5]. Furthermore, our study could find evidence to support the presumption by van Heesch et al., that the explicit documentation of responsibilities introduced by the stakeholder involvement viewpoint might cause architects to neglect the usage of stakeholder views, because they fear accountability [5].

5.5. Separation of Problem, Alternatives and Outcome

A result of RQ3 is that the decision model proposed by the framework is not well-suited for practice as the model does not separate the outcome of a decision from the background information, such as the underlying problem or the considered alternatives. Although the framework supports the concept of creating architecture backlog items by adding decisions as ideas, the fact that the underlying problem and outcome is embodied in the Architecture Decision entity (cf. Figure 1) and that alternatives are modeled as related Decision entities caused difficulties for the participants to model decisions. As a result, the architects asked for the possibility to express the problem, the alternatives and the outcome separately. This allows to explicitly identify problems that need to be addressed in the future. We have seen that this way of working is commonly used in the selected BUs. Hence, splitting the Decision entity into problem, alternatives and outcome would be a significant improvement for the Decision Viewpoint Framework and its tooling.

However, just adding more entities to the framework might not be the ideal solution as the framework focusses on simplicity. We have seen in the findings of RQ2 that complex approaches are not favored as they increase the learning curve and require more effort to document a decision. Hence, one of the questions for future research would be "How to integrate the concept of problem, alternative and outcome into the viewpoints and

the add-in?". A source of inspiration could be the decision meta-model proposed by Zimmermann et al. [41]. As illustrated in Figure 36, the model clearly separates the outcome of a decision from general information such as the problem.

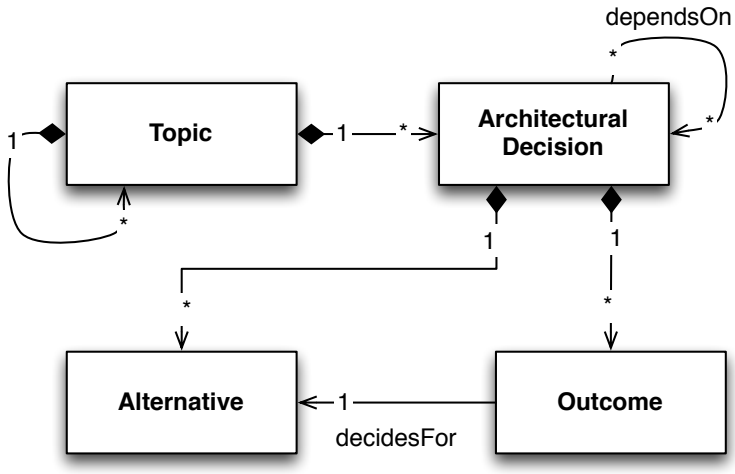


Figure 36: A simplified version of meta-model proposed by Zimmermann et al. [41].

5.6. Features and improvements of the add-in

The analysis of RQ3 showed that the tool is easy to use as long as it is closely integrated into the Enterprise Architect platform and adheres to its usage patterns. Minor points to improve usability could be identified. For instance, the tool does not integrate with the shortcut to create relationships between elements.

If we compare the developed add-in to the initially envisioned use cases, it can be seen that UC1[14], UC3[15] and UC4[16] are implemented. All five viewpoints are available in Enterprise Architect as diagrams or custom forms (i.e. Forces and Detail viewpoint). Information is synchronized between the viewpoints as decision entities are shared in the central repository. Thus state or name changes of decisions are immediately in effect across views. With regards to UC3 it is straightforward to create traces to existing modeling elements and to navigate traces from and to decisions. The developed add-in provides the ability to export decision documentation as Word, Powerpoint or Excel report. Depending on the target format, different information is included in the report or decisions are represented differently. However, UC2 – import decisions from other sources – was discarded during the requirement analysis in accordance with the stakeholder. As there is currently no standardized tool, there is also no standardized format for decisions thus increasing the complexity of the import function. A suitable solution to import unstructured AK was presented in the Knowledge Architect tool suite (cf. Section 1.2).

[14]Support the creation of decision views according to the five viewpoints
[15]Trace relationships between decisions and architecture design elements.
[16]Export decision documentation

Although the results of our exploratory study are promising and the tool was positively evaluated, there are many possibilities for improvements, such as the improvement of the decision model or the possibility to filter, group and collapse concerns in the Forces view. These suggestions for improvements and feature requests were incorporated in a backlog, which can be found in Appendix E.

6. Conclusion

Architecture decisions are often not explicitly documented in practice, partly because there are no decision documentation tools that are suitable for company-wide use. Therefore, we developed an extension to Enterprise Architect that facilitates the documentation of architecture decisions. The extension is based on a validated theoretical framework comprised of five architectural viewpoints.

In order to get further insights into the problem and to evaluate the developed tool, we conducted an exploratory case study with selected Business Units of ABB. The objective of the study was to assess the status quo of architecture decision documentation, to identify expectations of the ideal documentation approach, and to evaluate the advantages and limitations of the developed extension.

The findings of the study showed that although decision documentation receives rising awareness and decisions are documented within the Business Units, there is still a lack of a standard approach and no good tool-support. This makes it difficult to efficiently create and maintain decision documentation. We have also seen that architecture knowledge manifested in decisions is not reused or shared across projects.

With respect to the ideal tool, we identified that the ideal approach should focus on an efficient documentation and should be integrated into existing processes and tools. The possibility to extract stakeholder-specific reports from the captured decisions is an important use case of the tool.

The case-study showed that the tool is user-friendly and that the integration into Enterprise Architect is well done. For example, the ability to create traces between decisions and components has been desired for a long time. Providing multiple viewpoints to frame different concerns was appreciated by the architects as they could choose which viewpoint to use. We also received requests for new features and suggestions on how to improve the usability of the tool. However, a major conceptual limitation of the tool and theoretical framework is the inability to separate the outcome of a decision from the background information, such as the underlying problem or the considered alternatives.

From a tool development perspective, the items mentioned in the backlog should be implemented. From a research perspective, some of the questions that follow from the presented work are:

- How to further minimize the effort of documentation, e.g. by automating or generating parts of the documentation?
- How to make the reasoning process more efficient by reusing and sharing AK across projects?
- How to integrate the separation of problem, alternatives and outcome into the framework and the add-in?

In this project we gained further insights into the challenges and the context of decision documentation and evaluated a first version of the add-in. The project will continue

as it is planned to advance the tool and to conduct a second case-study to validate the improved add-in. The results of this project will help future tool development efforts to better understand the problems and the needs of practitioners with respect to decision documentation, so that decisions can be documented and used more effectively in a software process.

References

[1] U. van Heesch, P. Avgeriou, and R. Hilliard, "Forces on Architecture Decisions - A Viewpoint," in *Software Architecture (WICSA) and European Conference on Software Architecture (ECSA), 2012 Joint Working IEEE/IFIP Conference on*, 2012, pp. 101–110.

[2] "ISO/IEC/IEEE 42010 systems and software engineering — architecture description," Jan 2010.

[3] A. Jansen and J. Bosch, "Software Architecture as a Set of Architectural Design Decisions," in *WICSA '05: Proceedings of the 5th Working IEEE/IFIP Conference on Software Architecture (WICSA'05.* IEEE Computer Society, Nov. 2005, pp. 109–120.

[4] P. Kruchten, R. Capilla, and J. C. Dueñas, "The Decision View's Role in Software Architecture Practice," *IEEE Software*, vol. 26, no. 2, Mar. 2009.

[5] U. van Heesch, P. Avgeriou, and R. Hilliard, "A documentation framework for architecture decisions," *Journal of Systems and Software*, vol. 85, no. 4, Apr. 2012.

[6] J. Tyree and A. Akerman, "Architecture decisions: demystifying architecture," *Software, IEEE*, vol. 22, no. 2, pp. 19–27, 2005.

[7] J. S. van der Ven, A. Jansen, P. Avgeriou, and D. K. Hammer, "Using architectural decisions," in *Proceedings of the 2nd International Conference on the Quality of Software Architectures (QoSA).* Karlsruhe University Press, 2006, short paper.

[8] M. Babar, T. Dingsyr, P. Lago, and H. van Vliet, *Software Architecture Knowledge Management: Theory and Practice.* Springer, 2009.

[9] U. van Heesch, "Architecture Decisions: The next step Understanding, modeling, supporting and reviewing architecture decisions," Ph.D. dissertation, University of Groningen, Feb. 2013.

[10] C. Hofmeister, P. Kruchten, R. L. Nord, H. Obbink, A. Ran, and P. America, "A general model of software architecture design derived from five industrial approaches," *Journal of Systems and Software*, vol. 80, no. 1, pp. 106–126, 2007.

[11] P. Kruchten, "An ontology of architectural design decisions in software intensive systems," in *2nd Groningen Workshop on Software Variability*, 2004, pp. 54–61.

[12] L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice*, 2nd ed., ser. SEI Series in Software Engineering. Addison-Wesley, 2003, no. 0-321-15495-9.

[13] M. Shahin, P. Liang, and M. 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*, 2009, pp. 293–296.

[14] P. Liang and P. Avgeriou, "Tools and technologies for architecture knowledge management," in *Software Architecture Knowledge Management*, M. Ali Babar, T. Dingsøyr, P. Lago, and H. van Vliet, Eds. Springer Berlin Heidelberg, 2009, pp. 91–111.

[15] J. C. Dueñas and R. Capilla, "The decision view of software architecture," in *Software Architecture*. Springer, 2005, pp. 222–230.

[16] R. Capilla, F. Nava, and J. Dueas, "Modeling and documenting the evolution of architectural design decisions," in *Sharing and Reusing Architectural Knowledge-Architecture, Rationale, and Design Intent, 2007. SHARK/ADI'07: ICSE Workshops 2007. Second Workshop on*. IEEE, 2007, pp. 9–9.

[17] A. Tang, P. Avgeriou, A. Jansen, R. Capilla, and M. A. Babar, "A comparative study of architecture knowledge management tools," *Journal of Systems and Software*, vol. 83, no. 3, pp. 352 – 370, 2010.

[18] R. Farenhorst, P. Lago, and H. Van Vliet, "Effective tool support for architectural knowledge sharing," in *Software Architecture*. Springer, 2007, pp. 123–138.

[19] P. Liang, A. Jansen, and P. Avgeriou, "Knowledge architect: A tool suite for managing software architecture knowledge," University of Groningen, Tech. Rep., 2009.

[20] R. Capilla, F. Nava, S. Pérez, and J. C. Dueñas, "A web-based tool for managing architectural design decisions," *SIGSOFT Softw. Eng. Notes*, vol. 31, no. 5, Sep. 2006.

[21] L. Lee and P. Kruchten, "Customizing the capture of software architectural design decisions," in *Electrical and Computer Engineering, 2008. CCECE 2008. Canadian Conference on*, 2008, pp. 000 693–000 698.

[22] A. Jansen, J. Van Der Ven, P. Avgeriou, and D. K. Hammer, "Tool support for architectural decisions," in *Software Architecture, 2007. WICSA'07. The Working IEEE/IFIP Conference on*. Ieee, 2007.

[23] M. Babar and I. Gorton, "A tool for managing software architecture knowledge," in *Sharing and Reusing Architectural Knowledge - Architecture, Rationale, and Design Intent, 2007. SHARK/ADI '07: ICSE Workshops 2007. Second Workshop on*, 2007, pp. 11–11.

[24] R. Wieringa, "Design science as nested problem solving," in *Proceedings of the 4th international conference on design science research in information systems and technology*. ACM, 2009, p. 8.

[25] I. Sommerville, *Software Engineering*, eigth ed. Addison Wesley, 2006.

[26] A. Cockburn, *Writing effective use cases*. Addison-Wesley Reading, 2001, vol. 1.

[27] P. Kruchten, "Architectural blueprints—The "4+ 1" view model of software architecture," *IEEE Software*, vol. 12, no. 6, pp. 42–50, 1995.

[28] P. Avgeriou and U. Zdun, "Architectural patterns revisited – a pattern language," in *10th European Conference on Pattern Languages of Programs (EuroPlop 2005), Irsee*, 2005, pp. 1–39.

[29] G. Erich, H. Richard, J. Ralph, and V. John, "Design patterns: elements of reusable object-oriented software," *Reading: Addison Wesley Publishing Company*, 1995.

[30] R. K. Yin, *Case Study Research: Design And Methods (Applied Social Research Methods)*. Sage Publications, Inc, 2008, vol. 5.

[31] P. Runeson, M. Höst, A. Rainer, and B. Regnell, *Case Study Research in Software Engineering: Guidelines and Examples*. John Wiley and Sons, Inc., 2012.

[32] B. A. Kitchenham, S. Linkman, and D. Law, "DESMET: a methodology for evaluating software engineering methods and tools," *Computing & Control Engineering Journal*, 1997.

[33] J. M. Verner, J. Sampson, V. Tosic, N. A. A. Bakar, and B. A. Kitchenham, "Guidelines for industrially-based multiple case studies in software engineering," in *Research Challenges in Information Science, 2009. RCIS 2009. Third International Conference on*, 2009, pp. 313–324.

[34] P. Brereton, B. A. Kitchenham, and D. Budgen, "Using a protocol template for case study planning," in *Proceedings, 12th International Conference on Evaluation and Assessment in Software Engineering*, G. Visaggio, M. Baldassarre, M. Teresa, S. Linkman, and M. Turner, Eds., 2008.

[35] K. Pohl, *Requirements Engineering*. Springer, 2010.

[36] C. B. Seaman, "Qualitative Methods in Empirical Studies of Software Engineering," *IEEE Transactions on Software Engineering*, vol. 25, no. 4, pp. 557–572, July/August 1999.

[37] S. Adolph, W. Hall, and P. Kruchten, "Using grounded theory to study the experience of software development," *Empirical Software Engineering*, vol. 16, no. 4, Aug. 2011.

[38] A. Tang, M. Babar, I. Gorton, and J. Han, "A survey of the use and documentation of architecture design rationale," in *Software Architecture, 2005. WICSA 2005. 5th Working IEEE/IFIP Conference on*, 2005, pp. 89–98.

[39] M. Anvaari, R. Conradi, and L. Jaccheri, "Architectural decision-making in enterprises: Preliminary findings from an exploratory study in norwegian electricity industry," in *Software Architecture*. Springer Berlin Heidelberg, 2013, pp. 162–175.

[40] U. van Heesch and P. Avgeriou, "Mature architecting-a survey about the reasoning process of professional architects," in *Software Architecture (WICSA), 2011 9th Working IEEE/IFIP Conference on*. IEEE, 2011, pp. 260–269.

[41] O. Zimmermann, T. Gschwind, J. Küster, F. Leymann, and N. Schuster, "Reusable architectural decision models for enterprise application development," in *Software Architectures, Components, and Applications*, ser. Lecture Notes in Computer Science, S. Overhage, C. Szyperski, R. Reussner, and J. Stafford, Eds. Springer Berlin Heidelberg, 2007, vol. 4880, pp. 15–32.

[42] D. Garlan and M. Shaw, "An introduction to software architecture," *Advances in software engineering and knowledge engineering*, vol. 1, pp. 1–40, 1993.

[43] M. Shaw, "Larger scale systems require higher-level abstractions," in *ACM Sigsoft Software engineering notes*, vol. 14, no. 3. ACM, 1989, pp. 143–146.

[44] D. E. Perry and A. L. Wolf, "Foundations for the study of software architecture," *ACM SIGSOFT Software Engineering Notes*, vol. 17, no. 4, pp. 40–52, 1992.

[45] U. van Heesch, S. Mahdavi-Hezavehi, and P. Avgeriou, "Combining architectural patterns and software technologies in one design language," in *16th European Pattern Languages of Programming (EuroPLoP)*, Irsee, Germany, July 13-17 2011, proceedings.

[46] U. van Heesch, P. Avgeriou, and A. Tang, "Does decision documentation help junior designers rationalize their decisions? a comparative multiple-case study," *Journal of Systems and Software*, vol. 86, no. 6, pp. 1545–1565, Jun. 2013.

[47] U. Eklund and T. Arts, "A classification of value for software architecture decisions," in *Software Architecture*, ser. Lecture Notes in Computer Science, M. Babar and I. Gorton, Eds. Springer Berlin Heidelberg, 2010, vol. 6285, pp. 368–375.

[48] S. E. Hove and B. Anda, "Experiences from conducting semi-structured interviews in empirical software engineering research," *Software Metrics, 2005. 11th IEEE International Symposium*, p. 10, 2005.

[49] P. Kruchten, P. Lago, H. van Vliet, and T. Wolf, "Building up and Exploiting Architectural Knowledge," in *WICSA '05: Proceedings of the 5th Working IEEE/IFIP Conference on Software Architecture (WICSA'05*. IEEE Computer Society, Nov. 2005.

[50] V. Venkatesh and H. Bala, "Technology Acceptance Model 3 and a Research Agenda on Interventions," *Decision Sciences*, vol. 39, no. 2, pp. 273–315, May 2008.

[51] B. Kitchenham, L. Pickard, and S. L. Pfleeger, "Case studies for method and tool evaluation," *Software, IEEE*, vol. 12, no. 4, pp. 52–62, 1995.

[52] V. R. Basili, G. Caldiera, and H. D. Rombach, "The goal question metric approach," *Encyclopedia of software engineering*, 1994.

[53] A. Tang, Y. Jin, and J. Han, "A rationale-based architecture model for design traceability and reasoning," *Journal of Systems and Software*, vol. 80, no. 6, Jun. 2007.

Appendices

A. Life-cycle of a decision

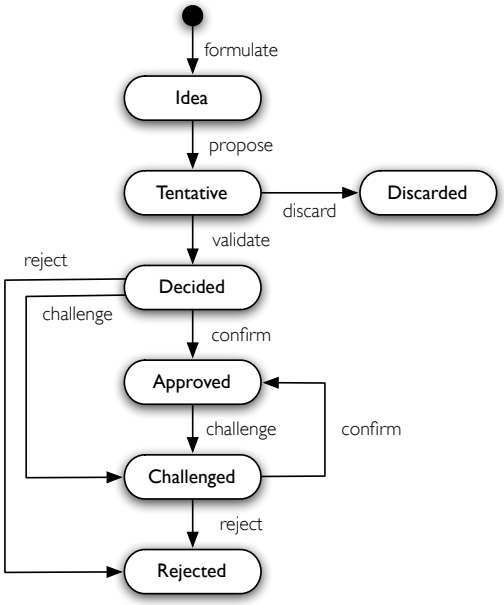


Figure 37: Life-cycle of a decision architecture decisions. Reproduced from [5].

B. Interview Guidelines

B.1. Requirements Elicitation Interview Guide

UC1: Five decision viewpoints. For each viewpoint let's discuss the following.

1. It's very important to keep the user interface simple. What is your feedback about the user interface in the demo?
2. How do you envision the tool to be integrated with your current EA workflow? Can you describe a typical use case?
3. How do you envision time savings when documenting decisions using the add-in tool?
4. Do you have suggestions to increase usability of the add-in tool support?
5. Do you have suggestions to increase efficiency of the add-in tool support?
6. When documenting the detailed decision viewpoint only the essential fields will be mandatory. Which fields do you consider essential according to your needs? The available fields are on slide 8 (decision detail viewpoint).

UC2: Import decision information from other sources. The tool will provide the possibility of importing decision information from other sources.

1. How are you currently documenting decisions? For example, which tools do you use (MS Word, Visio, Excel, Powerpoint)?
2. Which of these sources should be used for importing decision information in the EA? We can provide automatic import of content from structured sources.
3. Can you provide us with detailed example templates that you currently use for documenting decisions?
4. Should all the information found in the sources be imported? If not, then which is the essential information that should be imported?

UC3: Trace Relationships. The tool will provide capabilities to trace decisions to other architecture design elements.

1. Which design elements would you require traceability with decision elements?
2. Should the tracing between decisions and other design elements be bidirectional? For instance, from decisions to design elements and vice versa?
3. How would you like to indicate this relationship between decisions and other architecture design elements?
4. What would be the most efficient way for you to navigate through these relationships?

UC4: Export decision functionality.

1. Which document formats would you like to use for exporting decision documentation? (e.g. MS Word, PDF)
2. What information should be used when exporting viewpoint artifacts?
3. Is there a standard template to use for exporting viewpoint artifacts? If yes, can you please send us these templates?

Other questions

- Are there any other requirements for this project, within the scope of the proposal?
- What are the requirements regarding performance of the add-in?
- What are the requirements regarding security of the add-in? (e.g. authorization)
- What other non-functional requirements should we pay specific attention to?

B.2. Pre Tool Session Interview Guide

Step 1. Present Consent Information letter
Step 2. Introduction: discussion of interviewee?s work situation

Q1: Tell us a few words about your work (Domain, What kind of projects).

Q2: What are your roles and activities?

Q3: What about your experience (e.g. years in software architecture?)

Step 3. Status Quo in AD documentation

Q4: Roughly, in how many ADs are you involved (estimation per month, per year, per project)? By involve we mean participation as a decision-maker, or participation in a later stage (e.g. approving/rejecting decisions proposed by others).

Q5: Roughly, how many ADs do you document? (Percentage estimation: none, half, all) Per finite amount of time? Per project?

Q6: Roughly, how much time do you spend documenting ADs (half an hour, half a day, a week)?

Q7: What is your motivation for documenting ADs and are you obliged to document ADs?

Q8: What tools do you use for documenting ADs?

Q9: What processes/methods do you follow for documenting ADs?

Q10: Which stakeholders usually consume / read the documented ADs?

Q11: Have you experienced any limitations with your current way of documenting ADs?

Q12: How often do you read decisions that were documented by others and for which purposes?

a) In which form are they documented?

b) Are they comprehensible?

c) Do they contain sufficient information or do you miss certain aspects?

B.3. Post Tool Session Interview Guide

Step 1. Introduction: discussion of the first impression of the tool

Q1: Tell us about your first impression of the add-in.

- What would you suggest us to keep?
- What would you suggest to improve?

Q2: What is your first impression of the . . .

- Relationship view?
- Chronological view?
- Stakeholder involvement view?
- Forces view?
- Detail View?

Step 2. Perceived ease of use of the add-in

Q3: Tell us about how you use Enterprise Architect in your daily work. This question provides us with information about the architect?s familiarity with EA.

Q4: On a scale of 1-5(very easy), how easy to understand is the learning material?

Q5: On a scale of 1-5(sufficiently covers), how sufficiently does the learning material cover the functionality of the add-in?

Q6: On a scale of 1-5(very time-consuming), how time consuming do you consider documenting ADs using the add-in? Please motivate your answer.

Q7: Do you perceive the activity of using the add-in to be enjoyable? Do you prefer it to the ADD approaches that you had used in the past?

Q8: On a scale of 1-5(everything is clear), how well did you understand the tool based on our training session? Would you feel comfortable using the add-in on your own or would you require more training?

Q9: On a scale of 1-5(easy), how easy is it to create and configure a forces view? Please motivate your answer.

Q10: On a scale of 1-5(easy), how easy is it to create traces between decisions and other modeling elements (e.g. classes, components, or requirements)? Please motivate your answer.

Q11: Does the add-in feel like a natural part of Enterprise Architect?

Q12: In addition to what you already told us, what are the difficulties that you perceived while using the views?

Step 3. Perceived usefulness of the add-in

Q13: Does the add-in prevent you from unnecessary work?

Q14: Can you give us a rough cost-benefit estimate of using the add-in based on your personal evaluation?

Q15: Which stakeholders would appreciate the output of the add-in (managers, developers, customers)?

Q16: To what extent would you use the add-in during architecture decision documentation?

Q17: What would prevent you from using the current version of the add-in tomorrow? What are the factors that encourage or discourage you from using the add-in?

Q18: What would prevent you from using a "stable" version of the add-in tomorrow?

Q19: How valuable are the views produced by the add-in?

Step 4. Desired features, outlook and last words

Q20: Which features should exist in later versions of the add-in, and why?

Q21: Which features should be excluded from later versions of the add-in, and ?why?

Q22: Do you have any recommendations or things to improve?

C. Software Requirements Specification

1. INTRODUCTION 2
1.1. PURPOSE 2
1.2. BACKGROUND 3
1.3. DEFINITIONS, ACRONYMS, AND ABBREVIATIONS 4
1.3.1. Document-Wise 4
1.3.2. Sources in Requirements 4
1.4. OVERVIEW 4
2. OVERALL DESCRIPTION 4
2.1. PRODUCT PERSPECTIVE 4
2.2. PRODUCT FUNCTIONS 5
2.3. USER CHARACTERISTICS 5
2.4. CONSTRAINTS 5
2.5. ASSUMPTIONS AND DEPENDENCIES 5
2.6. APPORTIONING OF REQUIREMENTS 5
3. SPECIFIC REQUIREMENTS 6
3.1. FEATURES 6
3.1.1. Relationship Viewpoint 6
3.1.2. Chronological Viewpoint 6
3.1.3. Stakeholder Viewpoint 6
3.1.4. Detailed Viewpoint 6
3.1.5. Forces VP 7
3.1.6. Automation 7
3.1.7. Traceability 7
3.1.8. Word Export 8
3.1.9. PowerPoint Export 8
3.1.10. Excel Export 8
3.1.11. XMI Export 9
3.1.12. XMI Import (structured data) 9
3.1.13. Documentation / Help 9
3.2. USABILITY 9
3.3. EFFICIENCY 9
3.4. TESTING 9
3.5. SECURITY 9
3.6. PORTABILITY 9
3.7. EXCLUDED REQUIREMENTS 10
4. REFERENCES 11
5. APPENDIX 11
5.1. FIELDS OF THE DETAIL VIEWPOINT 11
5.2. DECISION TEMPLATE 12

1. Introduction

1.1. Purpose

This document describes the requirements for the *Decision Viewpoint Add-in for Enterprise Architect*, hereafter just called add-in.

The purpose of this document is the prioritization of requirements and reaching consensus on the add-in's functionality.

This document is intended for the Developers, Scrum Master, Product Owner, Customer (represented by Heiko), and ABB Architects (end users).

1.2. Background

Despite the benefits of decision documentation, architecture decisions are almost never explicitly documented in the industry. This is partly due to the fact that there is currently no decision documentation approach or tool that is suitable for use in industrial projects, in terms of required effort and user-friendliness.
Moreover, many existing decision documentation approaches do not integrate well with the rest of the architecture documentation (e.g. models, diagrams, views), as well as with the processes and tools that are used to design and document the architecture.

The documentation framework for architecture decisions [1] and its extension [2] enable the creation of intuitive decision models that can capture implicit architectural knowledge. It uses the conventions of the international standard for architecture description ISO/IEC/IEEE 42010 [3].

The decision framework consists of five viewpoints, each being dedicated to satisfying different stakeholder concerns related to architecture decisions.

- The *decision relationship viewpoint* makes relationships (e.g. causality, dependency) between architecture decisions explicit.
- The *stakeholder involvement viewpoint* explains the responsibilities of specific stakeholders in the decision-making process.
- The *decision chronological viewpoint* shows the evolution of architecture decisions over time.
- The *decision-forces viewpoint* makes relationships between decisions and decision forces explicit.
- Finally, the *decision detail viewpoint*, as an aggregate viewpoint, combines the information shown in all other views, by giving detailed information about single architecture decisions, including their rationale.

Each viewpoint is dedicated to framing specific decision-related concerns. At the same time, each viewpoint is integrated with the other viewpoints through a common meta-model to offer a more complete picture of decisions and their rationale.

We will develop an extension (add-in) to Sparx Systems' Enterprise Architect: a general-purpose model-driven, and extensible UML tool that can be used to create various types of models to support the architecting process.

The resulting add-in will allow documenting decisions in a user-friendly and efficient way, and seamlessly tracing them to other architectural design elements, like components. This will lead to a higher quality decision making process, better stakeholder communication, support for architecture reviews, and increased system evolvability.
To achieve the integration with other existing documentation, we will focus on traceability between architecture decisions and other architecture design

elements like components and connectors, bus technologies and configurations, or network design. The envisioned high level use cases of the add-in are described in detail in Section 2.2.

1.3.Definitions, acronyms, and abbreviations

This section provides the definitions, acronyms, and abbreviations used throughout this document for its proper interpretation.

1.3.1. Document-Wise

Add-in = Decision Viewpoint Add-In for Enterprise Architect
Element = refers to all EA model elements that are not decisions.
DV = Decision Viewpoints
VP = Viewpoint
EA = Enterprise Architect
MS = Microsoft

1.3.2. Sources in Requirements

T = Thomas
E = Emanuel
W = Wolfgang
H = Heiko
P = ABB Project Proposal, Decision Viewpoint Framework
A = Anonymous Architect who filled out the questionnaire
C = Christian

1.4.Overview

The document is structured according to IEEE 830 [4], which describes recommended practices for SRS. The rest of this document is organized as follows.
Section 2 provides add-in background information, such as its perspective and functions, constraints, and assumptions. These are useful to better understand the requirements documented in Section 3. This section consists of the functional requirements, structured based on the add-in's features, as well as the non-functional requirements.

2. Overall description

2.1.Product perspective

The add-in will function as part of EA's functionality. We will make sure that during implementation we will follow EA guidelines and best practices[1]. This will result in a seamless integration of the add-in with EA, without altering or losing any previous EA user experience, always in accordance with the add-in's functional requirements.

[1] http://www.sparxsystems.com/enterprise_architect_user_guide/

Our target version of EA is version 10. We will use the public API offered by EA and is defined in EA.Interop.dll. The target operating system is Windows 7.

2.2.Product functions

This section describes the high-level use cases of the add-in.

As a starting point, we envision the following high-level use cases for the add-in:

1. Support the creation of decision views according to the five decision viewpoints in a user-friendly and efficient way. Information that is specified in one of the views will automatically be propagated to the other views, so that no redundant information has to be provided by the user. Only the essential fields will be mandatory, while users will be able to select what information to fill in and what to skip. Emphasis will be given on keeping the user interface simple and reducing the learning curve.
2. Trace relationships between decisions and other architecture design elements. Such relationships can be established from both directions: decisions to design elements, and vice versa. This will apply both to direct and transitive relationships at configurable levels of depth. Examples of related design elements are components and connectors, deployment diagrams, or other types of models that can be created using Enterprise Architect.
3. Export decision documentation. The created decision views and their relationships with other design elements can be exported for use in other software tools, in order to create reports. The available formats (e.g. MS Office Word, or PDF) will be agreed upon with ABB. Reports will be generated based on templates; for example in the case of MS Word, a report template can be used that contains decisions views, their traces to other design elements and placeholders for explanatory text.

2.3.User characteristics

Architects, familiar with EA, not familiar with Decision Viewpoint framework

2.4.Constraints

Not using open-source. ABB can decide to make the add-in open source.
C#
Documentation in English

2.5.Assumptions and dependencies

2.6.Apportioning of requirements

Automatic import of unstructured decisions

3. Specific requirements

3.1.Features

3.1.1. Relationship Viewpoint

ID	Priority	Description	Source
RV1	Must	The add-in shall provide a toolbox that contains an item for each decision state.	H
RV2	Must	The add-in shall provide a toolbox that contains a connector for each relationship type.	H
RV3	Could	The add-in shall provide the option to display only decided, approved, challenged and idea decisions without displaying alternatives, rejected, or discarded decisions.	W
RV4	Must	The add-in should integrate with the EA Model Validation feature to validate relationship between decisions.	T
RV5	Could	The add-in shall provide a diagram that shows alternatives as a list and the decision would be one of those alternatives.	W
RV6	Must	The add-in shall provide a toolbox that contains a decision group.	D

3.1.2. Chronological Viewpoint

CV1	Should	The add-in should use baselines to store the decision history.	T
CV2	Should	The add-in shall automatically create a baseline to store the history, when a) The user creates a new iteration via the add-in menu b) Whenever a decision is created or modified c) On exit of Enterprise Architect, in case decisions have been changed (File Close)	C
CV3	Must	The add-in shall provide the possibility to manually add iterations, milestones and decisions to the chronological view.	P

3.1.3. Stakeholder Viewpoint

SV1	Could	The add-in shall visualize this viewpoint as table.	T
SV2	Must	The add-in shall visualize this viewpoint as diagram (as proposed in the paper).	T
SV3	Must	The add-in shall provide the possibility to manually add stakeholders, decisions, action connector, and iterations to the stakeholder involvement view.	P
SV4	Must	The add-in shall provide a toolbox that contains an action connector.	P
SV5	Must	The add-in shall provide a toolbox that contains a stakeholder element.	P
SV6	Could	The add-in shall integrate with the EA Model Validation feature to validate the model constraints.	P

3.1.4. Detailed Viewpoint

DV1	Should	The detail viewpoint shall aggregate the information from the other decision viewpoints.	P
DV2	Must	The add-in should display the detailed viewpoint in a separate window (Shall not reuse the default properties dialog of EA)	T
DV3	Could	The add-in should provide the option to reference external files as a separate field in the detail viewpoint.	E
DV4	Could	The add-in should provide the option to reference external resources via URI (e.g. SharePoint Items, Intranet documents) as a separate field in the detail viewpoint.	E
DV5	Must	The detailed viewpoint shall contain the following fields: Name, Version, Timestamp, State, Group, Problem/Issue, Decision, Alternatives, Arguments, Related Decision, Related Requirements, Related Design Elements, Related Files/URI, History, Additional Information	W
DV6	Must	The following fields are mandatory: Name and State	W
DV7	Could	The add-in shall check for completeness of decision documentation using the model validation feature of EA. Warning if a decision is missing one of the following fields: Problem/Issue, Decision, Alternatives, Arguments.	C
DV8	Must	The add-in should provide the possibility to open the decision detail view when the user double-clicks on a decision element. The default EA properties dialog is suppressed and only accessible via context menu.	E

3.1.5. Forces VP

FV1	Should	The add-in shall provide the possibility to select forces from a list. The list is populated with requirements that are defined in the EA project.	T, C
FV2	Must	The add-in shall provide the possibility to manually add forces and concerns.	T
FV3	Must	A manual added force should contain a description and a code.	P
FV4	Could	The add-in shall provide the possibility to select concerns from a predefined list of concerns.	C
FV5	Could	The add-in shall provide the possibility to extend the list of concerns.	C
FV6	Must	The add-in shall provide the possibility to add decisions and decisions group to a forces view.	P
FV7	Should	The add-in shall provide a wizard to create the structure of a forces view.	H, C
FV8	Could	The add-in shall provide a possibility to specify weights for each force.	H
FV9	Should	The add-in should integrate with the EA Model Validation feature to validate the constraints on the forces viewpoint.	

3.1.6. Automation

A1	Should	The add-in shall automatically update the forces view when a new decision is added to a decision group or removed from a decision group.	T
A2	Should	The add-in shall provide the possibility to automatically extract the history of decisions from EA baselines and add them to the chronological VP.	T
A3	Should	The add-in shall automatically update the stakeholder involvement view when a decision is created or modified.	T
A4	Should	The add-in shall update all decision viewpoints if a decisions is modified or created in one of other decision viewpoints.	C, P
A5	Should	The add-in shall automatically update the forces view if a decision alternative is added, modified or removed.	H

3.1.7. Traceability

T1	Must	The add-in shall integrate with the existing tracing capabilities of EA. For example, the Relationship Matrix, Trace-connectors and Impact Analysis View.	C
T2	Must	The add-in shall be able to create trace relationships between decisions and all structural and behavioral element types mentioned in the ABB modeling guidelines.	T, E, W
T3	Must	The add-in shall provide the possibility to trace decisions to elements.	T, E, W
T4	Could	The add-in shall provide the possibility to trace decisions to entire diagrams	W
T5	Should	The add-in shall extend the context menu of elements, so that it is possible to immediately create a decision that is traced to the respective element.	T
T6	Should	The add-in shall ask the user to update corresponding decisions if an element is modified.	T
T7	Must	The add-in shall provide the possibility to navigate from an element to the attached decision using the context menu.	T, E, W
T8	Must	The add-in shall provide the possibility to navigate from a decision to the attached elements using the context menu.	T, E, W
T9	Must	The add-in shall indicate trace relationships in the context menu of an element.	T
T10	Could	The add-in shall indicate the state of an attached decision by adjusting the border width and color of the element.	T
T11	Could	The add-in shall provide a traceability diagram that shows decisions and the related elements.	W

3.1.8. Word Export

WE1	Must	The add-in shall provide the possibility to export the decision viewpoints as MS Word documents (Office OpenXML). The generated document shall include all available decision views: Relationship, Stakeholder, Detail, Forces and the respective diagrams.	A, E, W
WE2	Could	The generated document shall include the trace relation between elements and decisions.	T, E
WE3	Could	The generated document shall use hyperlinks to navigate relationships and traces.	T
WE4	Could	The generated word document should contain a traceability matrix (overview).	T

3.1.9. PowerPoint Export

PE1	Could	The add-in shall provide the possibility to generate PowerPoint documents for Stakeholder communication, each slide showing the detail view of one decision.	T, H, E, W
PE2	Could	The add-in shall provide the possibility to select a set of decisions to be exported as PowerPoint document (Office OpenXML), each slide showing the detail view of one decision.	T, H, E, W

3.1.10. Excel Export

EE1	Could	The add-in shall provide the possibility to export forces views as Excel spreadsheets (Office OpenXML).	E

EE2	Could	The add-in shall provide the possibility to export the decision detail views in an Excel spreadsheet. Displaying the fields as columns and the values as rows.	W

3.1.11. XMI Export

XE1	Could	The add-in shall provide the possibility to export sub-trees of decisions as XMI.	H

3.1.12. XMI Import (structured data)

XI1	Could	The add-in shall provide the possibility to import sub-tress of decisions stored in the XMI format.	H

3.1.13. Documentation / Help

DH1	Could	Documentation and help of the add-in shall be accessible in the internal EA help system.	E
DH2	Must	Documentation shall be provided in the style of the ABB modeling guidelines (MS PowerPoint)	T
DH3	Should	Documentation shall be provided in form of video recordings that illustrate the main functionality	E
DH4	Could	Provide a public website to market the add-in and provide FAQs for quick learning.	H

3.2.Usability

U1	Must	The state of a decision should be clearly associated with a distinct color.	T
U3	Must	For an experience EA user, it shall be possible to use the add-in without prior training	W
U4	Must	Max menu depth <= 3 (Minimize number of mouse clicks)	W

3.3.Efficiency

E1	Must	The add-in should not make EA significantly slower.	T
E2	Should	The add-in shall be able to export 50 decisions in less than 10 minutes (PDF, Word, Excel, XMI).	T

3.4.Testing

T1	Should	The functionality of the add-in shall be tested with 50 decisions.	T

3.5.Security

S1	Must	The add-in shall integrate with the built-in EA security features.	T

3.6.Portability

P1	Must	The add-in shall not make use of deprecated APIs	H
P2	Must	The add-in shall support at least Sparx System Enterprise Architect v10	H
P3	Must	The add-in shall support Windows 7	H

3.7.Excluded Requirements

The following requirements were excluded due to technical limitations of Enterprise Architect or because they are beyond the scope of this project.

R07	The add-in should use the built-in version control feature of EA to store the decision history.	T

Enterprise Architect does not provide an API for the integrated Version Control functionality.

R35	The add-in should display markers on elements that indicate whether decisions are attached to this element or not.	T
R36	The add-in should display markers on elements that indicate inconsistencies among trace relationships.	T

Enterprise Architect does not provide a way to decorate or mark existing elements. The only possibility to change the appearance of elements is to add a custom stereotype. This might not be wanted as it alters the existing model and could lead to unforeseen side effects.

R38	The add-in shall indicate inconsistencies among trace relationships in the context menu of an element.	T
R45	The add-in should integrate with the EA Model Validation feature to validate the consistency of trace relations.	T

EA removes all relationships if an element is deleted. Hence, it is not possible to bring the model in an inconsistent state.

R40	The add-in shall provide the possibility to capture trace relationships across EA project files.	T

EA does not provide a way to trace/link elements across project files.

R41	The add-in should provide the possibility to trace decisions to requirements defined in SharePoint.	E
R42	The add-in should provide the possibility to trace SharePoint items to decisions in EA using links.	E
R43	The add-in may provide the possibility to automatically create SharePoint items from EA diagrams.	E
R60	The add-in shall provide the option to import decision information from SharePoint items.	

This would require a SharePoint installation for development. R41 is covered by R13, which allows to link resources by URI.

R47	The add-in should integrate with the existing export functionality used at ABB, i.e. Scribe tool.	T

Scribe is not used in all units of ABB.

R68	The add-in shall provide the possibility to export information in PDF format.	E

Word documents can be converted to PDF.

U02	The add-in shall provide keyboard shortcuts for frequently used context items

Enterprise Architect does not provide a possibility to add mnemonics or keyboard shortcuts. A custom implementation would be possible but this could lead to incompatibilities with existing (or future) shortcuts of EA.

R59	The add-in shall provide the possibility to import decisions from MS Word documents semi-automatically. The user shall be able to select and annotate text.

Importing decisions from unstructured documents is difficult. An implementation would be beyond the scope of this project.

4. References

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

[2] U. van Heesch, P. Avgeriou, and R. Hilliard. Forces on Architecture Decisions - A Viewpoint. In Proceedings of the Joint 10th Working IEEE/IFIP Conference on Software Architecture & 6th European Conference on Software Architecture, 2012.

[3] ISO/IEC/IEEE. Systems and software engineering — architecture description. ISO/IEC/IEEE 42010, pages 1–51, Jan 2010.

[4] IEEE Recommended Practice for Software Requirements Specifications, *IEEE Std 830-1998 DOI - 10.1109/IEEESTD.1998.88286*, 1998.

5. Appendix

5.1.Fields of the Detail Viewpoint

- Name
- Version
- Timestamp
- State
- Group
- Problem/Issue
- Decision
- Alternatives
- Arguments
- Related decisions
- Related requirements
- Related Elements

- Related Resources (Files/URI)
- History
- Additional Information

5.2.Decision Template

Decision template based on ISO/IEC/IEEE 42010:2012:

ID: AD-xx	Status: *idea, tentative, decided, approved, challenged, rejected, obsolesced*	Owner:	Timestamp: 2012-xx-yy
Name	*Short name for reference*		
Description	*Detailed description*		
Linked to system concerns	*Requirements id*		
Linked to architecture elements	*Components, interfaces, deployment nodes, quality scenarios, etc.*		
Rationale	*Why has this decision been made?*		
Constraints and assumptions	*e.g., business, technical constraints, other decisions*		
Alternatives	*Enumeration of considered alternatives*		
Consequences	*Any implications*		
Additional information	*e.g., standards, URLs, references to other documents*		

Architectural Recovery of the Repertory Grid Tool using Decisions

1. Scope and Purpose
    1.1 Stakeholders and Concerns
    1.2 Iterations
2. Forces
    2.1 Functional Requirements
    2.2 Quality Requirements
    2.3 Constraints
3. Views
    3.1 Decision Detail View
    3.2 Decision Chronological View
    3.3 Decision Relationship View

1. Scope and Purpose

RGT Version under consideration: 01.09.2011.
Status of this document: alpha

1.1 Stakeholders and Concerns

- SH-01: Project Manager
  (Dan)
  - Usability: motivation -> replace existing tool because it was less usable
    - Learnability: easy to create grids and interact
  - Costs -> No budget
  - Maintainability -> project is handed-over to other students
    - Documentation
  -
- SH-02: Software Engineers & Architects
  (Spyros, Jos, Grayson, Punitha, Dan)
  - Maintainability -> Show ability to create good software design for CV as.
  - Learning-experience: Project
  - Usability -> good design -> project becomes famous
    - Proper and sufficient user feedback
    - Sufficient “how to” information
  - Testability -> being able to easily write functional tests
- SH-03: Management
  (Paris)
    - Result of research group -> shows successful work
- SH-04: Researchers
  (Grayson, Dan)
  - Innovativeness -> Show some new aspects in order to publish papers
  - Evaluateability -> Case-studies can be performed to evaluate this tool
  -

- SH-05: End-Users
  - Reliability
    - Results are correct and comprehensible
  - Usability
    - Tool has no major bugs
    - Learnability
  -

1.2 Iterations

There were no milestones or iterations defined in this project. The following partitioning is based on a post-project evaluation of the project’s progress.

- **Phase 0** - 01/12 - 02/12
  - Spyros and Dan took initial architecture decisions
- **Phase 1** - 02/12 - 22/04/12
  - Grayson started implementation
  - Spyros was an advisory member of the project
- **Phase 2** - 23/04/12 - 11/05/12
  - Jos joined the project
  - Spyros actively implemented stuff
  - First alpha at PyGrunn
- **Phase 3** - 12/05/12 - 01/08/11
  - Finalize
- **Phase 4** - after August
  - Jos and Spyros left
  - Grayson stopped actively implementing stuff
  - Punitha joined

2. Forces

2.1 Functional Requirements

1. Collaboration
   a. Facilitator creates group session
      i. Sharing grids, creating a group session
   b. Facilitator inviting user to a group session
      i. Via email
      ii. Via token
2. Group session
   a. Facilitator can start/stop a session
   b. Showing grids of all participants per iteration
      i. Std dev, mean,
   c. Showing grids of all participants from previous iterations
   d. Specify session name
   e. Facilitator is able to consolidate grids of one iteration
   f. Facilitator can asks to revise weight/ratings or concerns/alternatives
3. Creating grids
   a. For group session

    b. For user
    c. Specify concerns
        i. At least 1
    d. Specify alternatives
        i. At least 2
    e. Specify grid name
    f. Undo all
4. Revise ratings/weights
    a. Adjust weight in a range between 1 and 100
    b. Adjust rating in a range between 1 and 5
    c. Show legend
5. Revise concerns/alternative
    a. Show legend
    b. Add or remove concerns
    c. Add or remove alternatives
6. Export dendrograms
    a. Facilitator can export dendrogram
7. Visualization of decisions using dendrograms
8. User account management
    a. Captcha
    b. Registration
    c. Email notification
    d. Change password
9. Interactive user interface

2.2 Quality Requirements

1. User gets the grid data in less than 1 second after requesting them.
2. User gets the sessions data in less than 1 second after requesting them.

2.3 Constraints

3. Views

3.1 Decision Detail View

Name	Self-hosted SVN
State	Rejected
Group	Project Management Tools
Problem/Issue	The project needs a central storage in order to work collaboratively. The storage should support versioning and access-control.
Decision	A self-hosted solution is used as a central storage.
Alternatives	No alternatives were considered.
Arguments	Dan had a shared-hosting server that could be used to host the subversion repository. A considered advantage was full-control over the SCM. Subversion has been chosen because of developers' experience.
Related Decisions	• Github <<replaces>> This

Related Requirements	
History	

Stakeholder	Action	Status	Iteration
Dan	Propose	Tentative	Phase 1
Grayson, Spyros	Validate	Decided	Phase 1
Dan	Reject	Rejected	Phase 2

Name	Github
State	Decided
Group	Project Management Tools
Problem/Issue	The project should be open-sourced in order to get more contributors. Therefore, we need to make the source code publicly available. This also requires more elaborated project management tools like an issue tracker.
Decision	Move the source code to Github and use the issue tracker on Github
Alternatives	1. Use Github for SCM and Mantis as issue tracker 2. Use Github for SCM and Trac as issue tracker
Arguments	Although Mantis (1) and Trac (2) are more elaborate project management tools, we use a simple issue tracker from Github. It is integrated with the source code repository and we do not need to spend more time to setup extra tools such as Mantis and Trac.
Related Decisions	• This <<replaces>> Self-hosted SVN • This <<caused by>> Open-source
Related Requirements	
History	

Stakeholder	Action	Status	Iteration
Dan	Propose	Tentative	Phase 2
Grayson, Spyros, Jos, Dan	Validate	Decided	Phase 2

Name	Open-source
State	Decided
Group	None
Problem/Issue	We want to attract more collaborators.
Decision	Make the project open-source.
Alternatives	No alternatives were considered.
Arguments	Involving more collaborators in the project can generate new ideas. This will also create a community from which valuable feedback can be received. This will also increase the productivity. However, as more developers join the project, the complexity of the management of who is doing what slightly increases, which might affect the progress of the project.
Related Decisions	• Github <<caused by>> This
Related Requirements	

History	

Stakeholder	Action	Status	Iteration
Dan	Propose	Tentative	Phase 2
Grayson, Spyros, Jos, Dan	Validate	Decided	Phase 2

Name	Python
State	Decided
Group	None
Problem/Issue	RGT should be a web-application hence a programming language is needed that facilitates web development.
Decision	Use Python
Alternatives	1. Scala 2. C# 3. PHP 4. Ruby
Arguments	Python is a language that can facilitate web development. There is a huge community and great support. However, also the rest of the alternatives (1), (2), (3), and (4) offer the same things. The main reason was that the engineering team was interested in acquiring development experience using Python. PHP (3) was also not considered because it has no clear separation between client and server implementation.
Related Decisions	• Django <<depends by>> This
Related Requirements	
History	

Stakeholder	Action	Status	Iteration
Spyros	Propose	Tentative	Phase 0
Grayson, Spyros, Dan	Validate	Decided	Phase 1

Name	Django
State	Decided
Group	None
Problem/Issue	Python itself has only limited features for web-application development. Hence, a web application framework is needed in order to ease development.
Decision	Use Django
Alternatives	1. Pylons 2. Web2py 3. TurboGears
Arguments	Django has a great community, which can offer support and feedback. It comes with extensive documentation and it offers native support of web development best practices and issues. Although, the rest of the alternatives (1), (2), and (3) satisfy these properties, the engineering team was keener to work with Django.
Related Decisions	• This <<depends on>> Python

Related Requirements	
History	

Stakeholder	Action	Status	Iteration
Spyros, Dan	Propose	Tentative	Phase 0
Grayson, Spyros, Dan	Validate	Decided	Phase 1

Name	SQLite
State	Decided
Group	None
Problem/Issue	We expect a variety of data from the users, so we need to store them.
Decision	Use SQLite as a database management system (DBMS).
Alternatives	No alternatives were considered.
Arguments	SQLite provides all the features that a modern relational DBMS should provide. Additionally, it is easy to setup and it facilitates development in the early stages.
Related Decisions	
Related Requirements	
History	

Stakeholder	Action	Status	Iteration
Grayson	Propose	Tentative	Phase 0
Grayson, Spyros, Dan	Validate	Decided	Phase 1

Name	MySQL for Production
State	Decided
Group	None
Problem/Issue	We need a DBMS for production. SQLite is not suited to be used in production mode.
Decision	Use MySQL for production.
Alternatives	1. PostgreSQL for production
Arguments	We also consider PostgreSQL (1) but we decide to use MySQL because it is a more widespread DBMS.
Related Decisions	
Related Requirements	
History	

Stakeholder	Action	Status	Iteration
Dan	Propose	Tentative	Phase 2
Grayson, Spyros, Jos, Dan	Validate	Decided	Phase 2

Name	South Migrations

State	Decided
Group	None
Problem/Issue	We need to migrate the database if models are changed.
Decision	Use South python package for migration.
Alternatives	No alternatives were considered.
Arguments	South provides a convenient way of migrating the database from old to new versions and vice versa in case of model changes.
Related Decisions	
Related Requirements	
History	

Stakeholder	Action	Status	Iteration
Spyros	Propose	Tentative	Phase 2
Dan, Grayson	Validate	Decided	Phase 2

Name	**jQuery**
State	Decided
Group	None
Problem/Issue	JavaScript is needed to create an interactive website. Writing plain JavaScript would be ineffective. Therefore, a JavaScript framework is needed that reduces time and effort.
Decision	Use jQuery.
Alternatives	No alternatives were considered.
Arguments	jQuery is a widely used JavaScript framework that provides all the desired functionality. Furthermore, the engineers feel more confortable of using this framework instead of considering alternatives.
Related Decisions	• jQuery UI <<caused by>> This
Related Requirements	
History	

Stakeholder	Action	Status	Iteration
Grayson	Propose	Tentative	Phase 1
Grayson	Validate	Decided	Phase 1

Name	**jQuery UI**
State	Decided
Group	None
Problem/Issue	We need to create interface widgets to provide a more interactive user interface. However, doing this using plain JavaScript will consume a lot of the project's available working hours and will decrease productivity.
Decision	Use jQuery UI.
Alternatives	No alternatives were considered.
Arguments	Using jQuery UI will increase productivity as it offers a widget library. It

	perfectly fits with jQuery and the engineering team is already familiar with jQuery.
Related Decisions	• This <<caused by>> jQuery
Related Requirements	
History	

Stakeholder	Action	Status	Iteration
Grayson	Propose	Tentative	Phase 1
Grayson, Spyros, Dan	Validate	Decided	Phase 1

Name	**Selenium**
State	Decided
Group	None
Problem/Issue	We need to implement functional testing. However, none of the current used technologies has native support for realizing functional testing.
Decision	Use Selenium.
Alternatives	No alternatives were considered.
Arguments	Selenium is a widely used framework for implementing functional tests. There is also a Python driver for Selenium, which makes it easier for the engineering team to work with it, as the project heavily depends on Python.
Related Decisions	
Related Requirements	
History	

Stakeholder	Action	Status	Iteration
Dan	Propose	Tentative	Phase 2
Grayson, (Dan)	Validate	Decided	Phase 2

Name	**Apache HTTPD**
State	Decided
Group	None
Problem/Issue	An application server is needed to deploy the tool for use in production.
Decision	Use Apache HTTPD as an application server.
Alternatives	1. NGinx
Arguments	We use Apache HTTPD because the engineering team had previous experience working with it. NGinx (1) was also considered for serving the static content but in the end the engineering team believed that this would introduce additional complexity.
Related Decisions	
Related Requirements	
History	

Stakeholder	Action	Status	Iteration

	Dan	Propose	Tentative	Phase 2
	Grayson, Spyros, Jos	Validate	Decided	Phase 2

Name	In-place Grid Creation
State	Decided
Group	None
Problem/Issue	The tool needs to provide a mechanism for grid creation. This should be implemented in an interactive and modern way.
Decision	Create an in place grid creation mechanism.
Alternatives	1. Google docs
Arguments	One alternative is to use Google docs (sheets) (1), which can be then integrated with the interface of the tool (e.g. as an iframe). However, we do not think that this would fit well with the style of the rest of the tool. Using our own in-place grid creation mechanism would let us customize it to fit our needs, without introducing extra unnecessary functionality.
Related Decisions	
Related Requirements	
History	

Stakeholder	Action	Status	Iteration
Dan	Propose	Tentative	Phase 1
Grayson, Spyros	Validate	Decided	Phase 1

Name	Wizard Grid Creation
State	Decided
Group	None
Problem/Issue	The in-place creation of grids proved to be quite complicated for new users. An alternative way is needed to facilitate the process of grid creation, especially for new users.
Decision	Add a wizard technique to create new grids.
Alternatives	No alternatives were considered.
Arguments	The wizard technique uses multiple steps and it guides the users during the creation of new grids. Guiding is provided with textual information.
Related Decisions	
Related Requirements	
History	

Stakeholder	Action	Status	Iteration
Dan	Propose	Tentative	Phase 3
Grayson, Spyros, Jos	Validate	Decided	Phase 3

Name	SVG Dendrograms
State	Decided
Group	None
Problem/Issue	Currently we create a png image and we show it in the user interface. This produces pixelated visual results, which are of low quality. We want to increase the quality of visualizing a dendrogram.
Decision	Visualize dendrograms using a vector based rendering technique (SVG).
Alternatives	No alternatives were considered.
Arguments	The advantage of using SVG is that we do not need to store them as we create them on the fly from the existing dendrogram data.
Related Decisions	
Related Requirements	
History	

Stakeholder	Action	Status	Iteration
Grayson	Propose	Tentative	Phase 2
Dan, Spyros, Jos	Validate	Decided	Phase 2

Name	SVG Export
State	Decided
Group	none
Problem/Issue	Users need to use the dendrograms for documentation purposes. It would be convenient for them to have a way to export the dendrograms.
Decision	Export dendrograms in SVG format.
Alternatives	1. PDF Export 2. PNG Export
Arguments	Currently the dendrograms are displayed in the web page using SVG. So from the implementation perspective it is much more productive to use SVG export. Plus the quality of the SVG images is very high. For the other two alternatives (1) and (2) we would need to introduce a lot of extra code and more research would needed, which would decrease productivity at that particular time.
Related Decisions	• This <<caused by>> SVG Dendrograms
Related Requirements	
History	

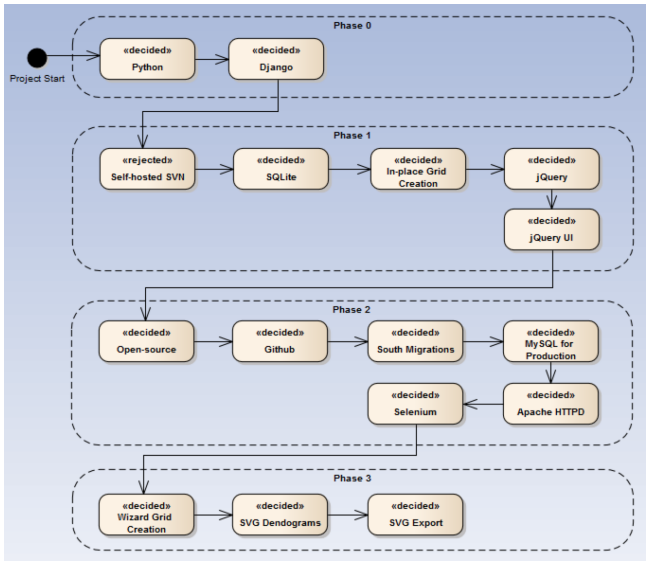
Stakeholder	Action	Status	Iteration
Grayson	Propose	Tentative	Phase 3
Dan, Spyros, Jos	Validate	Decided	Phase 3

Name	Source Code Decomposition according to concerns
State	Decided
Group	none

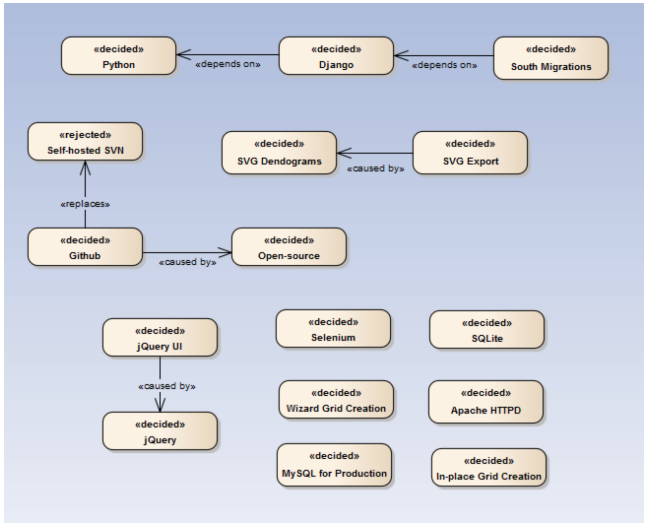
Problem/Issue	The project has various functionalities, each one addressing different concerns, and they need to be separated.
Decision	Decompose the source code in modules according to concerns. Each module addresses a separate concern.
Alternatives	No alternatives were considered.
Arguments	The source code is easily maintainable as far as the files' structure is concerned.
Related Decisions	
Related Requirements	
History	

Stakeholder	Action	Status	Iteration
Grayson	Propose	Tentative	Phase 1
Spyros	Validate	Decided	Phase 1

3.2 Decision Chronological View



3.3 Decision Relationship View



Backlog Items

The backlog item list comprises of feature requests/bugs emerged from the case study interviews, and requirements from the requirements specification document, which have not yet been addressed.

General

Backlog Item	Priority	Time Estimation	Req. ID
Change the metamodel of decision viewpoints elements as defined in the JSS paper, to support the notion of Issue-Alternative-Outcome. Many architects identified the problem of not being able to represent a decision topic or issue. Furthermore, they identified that modelling alternatives as separate decisions creates a lot of confusion.	Must	4 weeks	
Give automatically the viewpoint name to new diagrams. For example, when a relationship diagram is created, give it the name Relationship.	Could	1 day	
Investigate the possibility to create new decision states. Although this is doable, it is perhaps too dangerous. Adding even more states might overcomplicate things.	Could	< 1 week	
Provide possibility to show/hide specific type of connectors in the diagrams.	Must	< 1 week	
A date (architect does not specify either modified or created date?) should be visible on decision elements in diagrams.	Could	< 1 week	
Use the arrow that appears in the top-right of the elements when mouse-over them, to quickly create new elements or connectors.	Could if possible	< 1 week	
The add-in shall check for completeness of decision documentation using the model validation feature of EA. Warning if a decision is missing one of the following fields: Problem/Issue, Decision, Alternatives, Arguments.	Could	< 1 week	DV7

Automation

Backlog Item	Priority	Time Estimation	Req. ID
Generate other Viewpoints out of the Detail Viewpoint (for example the architect could add history manually, and other views could be auto generated).	Could	4 weeks	

Relationship

Backlog Item	Priority	Time Estimation	Req. ID
The add-in shall provide the option to display only decided, approved, challenged and idea decisions without displaying alternatives, rejected, or discarded decisions.	Could	< 1 week	RV3
The add-in shall provide a diagram that shows alternatives as a list and the decision would be one of those alternatives.	Could	< 2 weeks	RV5

Chronology

Backlog Item	Priority	Time Estimation	Req. ID
Filter specific states in the chronology viewpoint (for example see only the decisions with state decided).	Must	< 2 weeks	
Filter specific decisions or group of decisions in the chronology viewpoint.	Must	< 2 weeks	

Forces

Backlog Item	Priority	Time Estimation	Req. ID
The add-in shall provide the possibility to select forces from a list. The list is populated with requirements that are defined in the EA project.	Should	< 1 week	FV1
A manual added force should contain a description and a code.	Must	1 day	FV3
The add-in shall provide the possibility to	Could	< 1 week	FV4

select concerns from a predefined list of concerns.			
The add-in shall provide the possibility to add decision groups to a forces view.	Should	< 2 weeks	
The add-in shall provide a possibility to specify weights for each force.	Could	< 1 week	FV8
The add-in shall integrate with the EA Model Validation feature to validate the constraints on the forces viewpoint.	Could	< 2 weeks	FV9
The add-in shall provide the possibility to write additional text (also mentioned as notes) in the rating fields.	Must	< 2 weeks	
The add-in shall the possibility to expand/collapse (also referred as hiding) groups of concerns.	Should	< 2 weeks	
The add-in shall provide the possibility to prioritize concerns using hundred-dollar.	Should	< 2 weeks	
The add-in shall provide the possibility to add confidence factors (as probabilities) to ratings.	Should	< 2 weeks	
The add-in shall provide he possibility to drag and drop requirements and decisions on the forces viewpoint directly.	Could	< 2 weeks	
The add-in shall provide the possibility to hide decisions.	Should	< 1 week	
Implement colouring in forces viewpoint for the cells.	Should	< 2 weeks	
The add-in shall provide the possibility to sort rows in forces viewpoint.	Should	1 day	
Synchronize Excel with forces viewpoint.	Could	> 2 weeks	
The add-in shall provide the possibility to select different evaluation schemas (5-point scale, 3-point scale, numbers, free-text).	Must	< 2 weeks	
The add-in shall provide a drop down menu in rating fields to select rating.	Must	< 1 week	

Stakeholder Involvement

Backlog Item	Priority	Time Estimation	Req. ID
Show the modified date of the "action" connector in the stakeholder involvement viewpoint.	Could	< 1 week	
The add-in shall visualize this viewpoint as table.	Could	> 2 weeks	RV1

Detail

Backlog Item	Priority	Time Estimation	Req. ID
Show the links to architecture elements as list.	Should	1 day	
Add field "Consequences".	Should	1 day	
Add field "Additional Information".	Should	1 day	
The add-in should provide the option to reference external files as a separate field in the detail viewpoint.	Could	1 day	DV3
The add-in should provide the option to reference external resources via URI (e.g. SharePoint Items, Intranet documents) as a separate field in the detail viewpoint.	Could	1 day	DV4
The following fields are mandatory: Name and State.	Must	1 day	DV6

Export

Backlog Item	Priority	Time Estimation	Req. ID
The generated document shall include the trace relation between elements and decisions.	Could	< 1 week	WE2
The generated document shall use hyperlinks to navigate relationships and traces.	Could	< 1 week	WE3
The generated word document should contain a traceability matrix (overview).	Could	< 1 week	WE4
The add-in shall provide the possibility to select a set of decisions to be exported as PowerPoint document (Office OpenXML), each slide showing the detail view of one decision.	Could	< 1 week	PE2
The add-in shall provide the possibility to export the decision detail views in an Excel spreadsheet. Displaying the fields as columns and the values as rows.	Could	< 1 week	EE2

Traceability

Backlog Item	Priority	Time Estimation	Req. ID
Give semantic meaning to trace relationship, such as "triggers".	Could	< 1 week	
The add-in shall provide the possibility to trace decisions to entire diagrams.	Could	< 2 weeks	T4
The add-in shall indicate the state of an	Could	< 1 week	T10

attached decision by adjusting the border width and colour of the element.			

Documentation

Backlog Item	Priority	Time Estimation	Req. ID
Documentation shall be provided in the style of the ABB modelling guidelines (MS PowerPoint)	Must		DH2
Documentation shall be provided in form of video recordings that illustrate the main functionality.	Should		DH3
Provide a public website to market the add-in and provide FAQs for quick learning.	Could		DH4