

TECHNISCHE UNIVERSITÄT MÜNCHEN

Lehrstuhl für Informatik XIX

**Tool support for architectural decision making in large  
software intensive projects**

Manoj Mahabaleshwar

Vollständiger Abdruck der von der Fakultät für Informatik der Technischen Universität München  
zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Vorsitzender: Univ.-Prof. Dr. Alexander Pretschner

Prüfer der Dissertation:

1. Univ.-Prof. Dr. Florian Matthes
2. Univ.-Prof. Dr. Bernd Brügge

Die Dissertation wurde am 31.10.2019 bei der Technischen Universität München  
eingereicht und durch die Fakultät für Informatik am 08.03.2020 angenommen.



# Zusammenfassung

Während des letzten Jahrzehnts hat sich ein Paradigmenwechsel vollzogen, in der Weise, in der Wissenschaftler und Praktiker Softwarearchitekturen betrachten. Eine Softwarearchitektur wird heutzutage nicht nur als Blaupause gesehen, sondern als eine Menge von Designentscheidungen, die zur Blaupause des zugrundeliegenden Softwaresystems führen. Architekten und Entwickler treffen regelmäßig Designentscheidungen, die zum Beispiel die Wahl eines Architekturstils, eines Entwurfsmusters und der zugrundeliegenden IT-Infrastruktur betreffen. Die Dokumentation solcher Entscheidungen hilft, in großen Projekten, Fragen zu beantworten wie: „Welche Designentscheidungen wurden bereits getroffen?“, „Welche Architekturelemente und Qualitätsattribute werden von einer Entscheidung beeinflusst?“, „Wer sollte in eine neue Entscheidung involviert sein?“, „Welche ähnlichen Entscheidungen wurden in der Vergangenheit getroffen?“ und „Welche Alternativen sollten beim Treffen einer Entscheidung berücksichtigt werden?“.

Obwohl zahlreiche Tools zum Architektur-Wissensmanagement (architectural knowledge management, AKM) das Dokumentieren von Designentscheidungen unterstützen, beobachten wir, dass diese Entscheidungen in der Praxis selten dokumentiert werden. Die Gründe warum Designentscheidungen nicht explizit dokumentiert werden sind zahlreich. Diese umfassen, ohne darauf beschränkt zu sein, die fehlende Integration von AKM Tools in der Software Engineering Praxis. Darüber hinaus werden solche AKM Tools als störend wahrgenommen, da sie den Architekten dazu zwingen, Designentscheidungen manuell zu dokumentieren, was umständlich und teuer ist.

Im Gegensatz zu traditionellen AKM Tools, die einen Top-Down-Ansatz verfolgen, schlagen wir einen Bottom-Up-Ansatz für das Kuratieren und die Wiederverwendung von Architekturwissen vor. Im Bottom-Up-Ansatz werden Architekturentscheidungen automatisch aus verschiedenen Systemen abgerufen und mit öffentlich verfügbaren Wissensquellen erweitert. Spezifisch um Designentscheidungen die in der Vergangenheit getroffen wurden zu verstehen und um Architekten beim Treffen zukünftiger Entscheidungen zu unterstützen, präsentieren wir die Umsetzung eines Amelie - Decision Explorer (ADeX) genannten Frameworks. Die Komponenten innerhalb des Frameworks rufen zunächst projektbezogene Informationen von verschiedenen Systemen ab und identifizieren dann automatisch Designentscheidungen innerhalb der Projektdaten unter Verwendung von Techniken des maschinellen Lernens.

Im nächsten Schritt annotiert ADeX, mit Hilfe von Natural Language Processing, Architekturelemente und Qualitätsattribute, die von den extrahierten Entscheidungen beeinflusst werden. Unter Verwendung der Annotationen aus dem vorherigen Schritt quantifiziert ADeX die Architektur-Expertise von Individuen, um Experten für die Lösung neuer Designprobleme zu empfehlen. Schließlich schlägt ADeX, unter Verwendung eines Ontologie-basierten Ansatzes, Alternativen vor, die beim Treffen von Architekturentscheidungen (architectural decision-making, ADM) berücksichtigt werden können. Basierend auf den Ergebnissen dieser Schritte werden verschiedene Standpunkte generiert, um Architekten bei der Beantwortung der zuvor genannten Fragen im ADM-Prozess zu unterstützen.





---

Die qualitative Evaluation von ADeX zeigt, dass es den benötigten Aufwand zur Pflege von Architekturwissen reduziert und Stakeholdern einen Rückblick auf Designentscheidungen in großen Softwareprojekten ermöglicht. Darüber hinaus wurden die Komponenten innerhalb des Frameworks quantitativ evaluiert, hierfür wurden Datensätze aus Open-Source- und Industrieprojekten verwendet. Basierend auf diesen Datensätzen fanden wir heraus, dass Entscheidungen mit einer Genauigkeit von 91% automatisch aus Issues extrahiert werden können, Architekturelemente können mit einer Genauigkeit von 84% annotiert werden und Experten können mit einer Genauigkeit von 79% empfohlen werden um neue Designprobleme zu adressieren.

Außerdem wird ADeX, das während der letzten vier Jahre realisiert wurde, aktuell im Rahmen einer AI4AM genannten Initiative von unserem Industrie-Partner verwendet, um seine Vorteile für die Geschäftsbereiche hervorzuheben. Ein solches Setup bietet uns die nötige Plattform um verwandte Herausforderungen im ADM zu untersuchen, inklusive der Aspekte Bias und Unsicherheiten beim Treffen von Entscheidungen.



# Abstract

Over the last decade, there has been a paradigm shift in the way researchers and practitioners view software architectures. Today, a software architecture is not only seen as a blueprint but instead is considered as a *set of design decisions* that leads to the blueprint of the underlying software system. Architects and developers regularly make design decisions, for instance, concerning the selection of an architectural style, design pattern, and also the underlying IT infrastructure. Documenting such decisions especially in large projects is beneficial to answer questions such as “*What design decisions have already been made?*”, “*Which architectural elements and quality attributes are affected by a decision?*”, “*Who should be involved in making a new decision?*”, “*Which similar decisions have been made in the past?*”, and “*What are the alternatives to consider while making a design decision?*”.

Even though many of the architectural knowledge management (AKM) tools provide means to capture design decisions, we observe that, in practice, those *decisions are rarely documented*. The reasons for not explicitly capturing design decisions are manifold. These include, but not limited to, the lack of integration of AKM tools in software engineering practice. Second, these AKM tools are considered intrusive since they require architects to manually document decisions which can be tedious and expensive.

Contrary to the traditional AKM tools that follow a top-down approach, we propose a *bottom-up approach for AK curation and reuse* thereof. In the bottom-up approach, design decisions are automatically retrieved from various systems as well as annotated with publicly available knowledge sources. Specifically for understanding design decisions made in the past and for supporting architects in making future decisions, we present the realization of a framework called *Amelie - Decision eXplorer (ADeX)*. The components within the framework first retrieve project-related information from disparate systems and then, automatically identify design decisions within project’s data using machine-learning techniques. Next, by using natural language processing, ADeX annotates architectural elements and quality attributes affected by the extracted decisions. Using the annotations from the previous step, ADeX quantifies the architectural expertise of individuals to recommend experts for addressing new design concerns. And finally, using an ontology-based approach, ADeX suggests alternatives that can be considered during architectural decision-making (ADM). Based on the results of these steps, various viewpoints are generated to support architects answer the aforementioned questions during the ADM process.

The qualitative evaluation of ADeX shows that it reduces the effort required for maintaining AK and provides stakeholders with a retrospective view of design decisions in large software projects. Furthermore, the components within the framework have been quantitatively evaluated using datasets both from open-source and industrial projects. Based on those datasets, we found that decisions can be automatically extracted from issues with an accuracy of 91%, architectural elements can be annotated with an accuracy of 84%, and with an accuracy of 79% experts can be recommended to address new design concerns.

Finally, ADeX, which has been realized over the last four years is currently being used by our industry partner to highlight its benefits to the business units under an initiative called AI4AM. Such a setup provides us the necessary platform to further investigate related challenges of ADM including the aspects of biases and uncertainties in decision making.



# Acknowledgment

For writing this acknowledgment section, when I reflect on how my journey began at the Software Engineering for Business Information Systems (sebis) Chair, it is hard to believe that I have been associated with the sebis chair for almost seven years now; first, as a master's student and then, as a PhD candidate. My first interaction with Prof. Dr. Florian Matthes in 2012, when I approached him for a working student position remains vivid in my memory. His enthusiasm during the discussions, openness, and encouragement for new ideas as well as playing the devil's advocate to challenge my train of thoughts have remained consistent over the years. For providing me the opportunity and the environment to grow as a researcher, for supervising my dissertation, and for guiding and encouraging me during difficult times in the process with extreme patience, my utmost gratitude goes to my Doktorvater - Prof. Dr. Florian Matthes. I also want to thank Prof. Dr. Bernd Brügge for the discussions related to my dissertation and for being my second reviewer. I specifically thank him for pointing out the missing references in the related literature, for reminding me to include the threats to validity in the evaluation section, and for suggesting a better structure to the introduction chapter.

At the sebis chair, collaborations and discussions with my colleagues not only contributed to co-authored publications and successful execution of lectures but more importantly, helped me grow as a researcher by broadening my limited understanding of many topics including enterprise architecture management, model-based software systems, natural language processing and understanding, machine learning, software ecosystems, and blockchain. Special thanks to Klym Shumaiev for his contributions to our joint research ventures in the area of software architecture knowledge management and related projects with our industry partner. I would also like to acknowledge the efforts of Daniel Braun for translating the abstract of this dissertation into the German language. Many thanks to all the students who contributed to my research with their master's and bachelor's theses.

I would like to thank my colleagues in the Architecture Definition and Management department at Siemens Corporate Technology. Especially, Andreas Biesdorf and Uwe Hohenstein who have provided valuable inputs to my research and have helped me with organizational issues, such as, getting contacts with business units at Siemens to conduct the case studies in my research.

Lastly, I would like to express my gratitude to my family for their continued support. I am grateful to my parents for their understanding and for encouraging me during my dissertation. Geographically being miles apart, throughout my dissertation, they made sure that I was not distracted from my goals even during difficult times. Finally, during the last phase of my dissertation, my significant other Deepa Hegde made sure that my social commitments were met through her efforts. I am thankful to her for her patience, her encouragement, and for her persistent reminders that I need to complete the final submission process of my dissertation.

Garching b. München



Manoj Mahabaleshwar



---

## Table of Contents

---

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Problem statement and research questions . . . . .	3
1.2	Contributions of this dissertation . . . . .	6
1.3	Outline of this dissertation . . . . .	8
<b>2</b>	<b>Foundations and related work</b>	<b>9</b>
2.1	Research methodology for conducting the semi-systematic literature review . . . . .	9
2.1.1	Inclusion and exclusion criteria for identifying research publications . . . . .	10
2.1.2	Search strategy for identifying relevant research publications . . . . .	11
2.1.3	Data extraction and synthesis for conducting the literature review . . . . .	12
2.2	Review results of the identified 227 publications . . . . .	13
2.2.1	Architectural knowledge management . . . . .	14
2.2.2	Architectural design decisions . . . . .	18
2.2.3	Architectural design rationale . . . . .	31
2.2.4	Architectural decision making . . . . .	32
2.2.5	Group decision making . . . . .	38
<b>3</b>	<b>Requirements elicitation</b>	<b>41</b>
3.1	Requirements related to architectural knowledge management . . . . .	42
3.2	Requirements related to architectural decision making . . . . .	45
<b>4</b>	<b>A conceptual framework for architectural decision making</b>	<b>49</b>
4.1	Stakeholders of the framework . . . . .	50
4.2	Components and their responsibilities within the framework . . . . .	51
4.2.1	SocioCoretx: A meta-model based AKM system . . . . .	52
4.2.2	SyncPipes: A data integration and synchronization platform . . . . .	53
4.2.3	Decision classifier: A machine-learning based document classifier . . . . .	54
4.2.4	Architectural elements annotator: A named-entity extractor . . . . .	54

4.2.5	Architectural solutions recommender: An ontology-based approach . . . . .	55
4.2.6	Rationale extractor: Identifying the rationale behind design decisions . . . . .	56
4.2.7	Expert recommender: User-profile based recommendations . . . . .	57
4.2.8	Workbench4DC: Clustering similar design decisions . . . . .	57
4.2.9	Amelie - Decision Explorer: User interfaces for end-users . . . . .	58
4.3	Process steps within the AKM framework . . . . .	59
<b>5</b>	<b>System design and implementation</b>	<b>61</b>
5.1	SocioCortex . . . . .	62
5.1.1	The hybrid-wiki meta-model . . . . .	63
5.1.2	Using SocioCortex for architectural knowledge management . . . . .	64
5.1.3	The dynamic architectural knowledge model . . . . .	65
5.1.4	The static architectural knowledge model . . . . .	66
5.1.5	User interface for capturing architectural knowledge in SocioCortex . . . . .	69
5.2	SyncPipes . . . . .	71
5.2.1	The configuration of the extractor and the loader services in SyncPipes . . . . .	72
5.2.2	Data transformation using user-defined data mappings . . . . .	74
5.2.3	User interfaces for managing data transformation . . . . .	76
5.3	Document classifier . . . . .	79
5.4	Workbench4DC: Document clustering component . . . . .	82
5.5	Akre-Server: Architectural recommendations component . . . . .	87
5.5.1	Architectural solutions recommender . . . . .	88
5.5.2	Rationale extractor . . . . .	96
5.5.3	Expert recommender . . . . .	98
5.5.4	System design of the Akre-Server component . . . . .	101
5.6	Amelie - Decision explorer client . . . . .	103
<b>6</b>	<b>Evaluation</b>	<b>111</b>
6.1	Quantitative evaluation of components within ADeX . . . . .	111
6.1.1	Quantitative evaluation of the architectural annotator component . . . . .	112
6.1.2	Quantitative evaluation of the decision classifier component . . . . .	115
6.1.3	Quantitative evaluation of the expert recommender . . . . .	124
6.2	Qualitative evaluation of ADeX in real-world projects . . . . .	131
6.3	Evaluation summary . . . . .	137
<b>7</b>	<b>Future work and conclusion</b>	<b>139</b>
7.1	Lessons learned . . . . .	139
7.2	Ongoing research activities . . . . .	141
7.3	Conclusion . . . . .	148
	<b>Bibliography</b>	<b>151</b>
	<b>Primary studies used in literature review</b>	<b>162</b>
	<b>Abbreviations</b>	<b>181</b>



---

## List of Figures

---

1.1	The representation of ADD and its associated concepts in an ADD model. The concepts in the ADD model have been derived from [1–4] (cf. Section 2.2.2.1 for a detailed discussion) . . . . .	3
1.2	Outline of this dissertation . . . . .	8
2.1	Planning, conducting, and reporting a systematic literature review: as prescribed by Kitchenham and Charters [5] . . . . .	10
2.2	The search strategy that resulted in 227 publications for conducting the semi-systematic literature review . . . . .	12
2.3	The trend in the number of publications related to AKM, ADDs, and ADM distributed over the past years . . . . .	13
2.4	The classification of the identified 227 publications into different AK categories . . . . .	14
2.5	A traceability support system . . . . .	18
2.6	The context of CBAM . . . . .	19
2.7	The conceptual model of ADD used in the Archium approach . . . . .	22
2.8	A conceptual model capturing ADD and its associated concepts . . . . .	26
2.9	The conceptual model of the architecture design decision support system . . . . .	27
2.10	An architectural design rationale model by Burge and Brown . . . . .	31
4.1	A conceptual framework for managing ADDs and supporting the ADM process . . . . .	50
4.2	The realization of a conceptual framework for managing ADDs and supporting ADM . . . . .	52
4.3	The process steps within ADeX to support the ADM process: using the Apache Spark project as an example scenario . . . . .	60
5.1	The high-level architecture of ADeX: the core components and their dependencies . . . . .	62
5.2	The hybrid-wiki meta-model of SocioCortex: image taken from [6] . . . . .	63
5.3	The design of an AKM system using SocioCortex as the backend . . . . .	64

5.4	The dynamic AK model for capturing the design and the context knowledge (the concepts have been taken from [7–11]) . . . . .	66
5.5	The static AK model for capturing general AK . . . . .	67
5.6	An exemplary MxL expression for finding domain experts based on an ECA rule . . . . .	69
5.7	The user interface of SocioCortex for collaboratively modeling the application domain and for capturing the instances of the domain concepts . . . . .	70
5.8	The conceptual model of SyncPipes for AK integration and synchronization . . . . .	71
5.9	Interfaces for implementing the extractor and loader services in SyncPipes . . . . .	72
5.10	Concepts for persisting ETL pipelines and data mappings in SyncPipes . . . . .	73
5.11	The steps in the data transformation workflow in SyncPipes . . . . .	74
5.12	Sequence diagram showing the data extraction and loading process in SyncPipes . . . . .	75
5.13	The dashboard view in SyncPipes shows the available extractor and loader services as well as the status of the ETL jobs . . . . .	76
5.14	The user interface for configuring an extractor or a loader service in SyncPipes . . . . .	77
5.15	The user interface for configuring a pipeline’s data mapping in SyncPipes . . . . .	78
5.16	The user interface for editing and executing a pipeline in SyncPipes [source: [12]] . . . . .	78
5.17	The high-level overview of the design decision detection and classification process . . . . .	79
5.18	The high-level system design of the document classification component . . . . .	80
5.19	A class diagram showing the concepts and their relationships in the document classification component . . . . .	81
5.20	A machine learning pipeline for design decision detection and classification . . . . .	81
5.21	The application controllers and their dependencies for managing business logic in the document classification component . . . . .	82
5.22	The high-level overview of the document clustering process . . . . .	83
5.23	The high-level system design of the document clustering component . . . . .	83
5.24	A class diagram showing the concepts and their relationships in the document clustering component . . . . .	84
5.25	The user interface for creating and configuring a document clustering pipeline . . . . .	85
5.26	The user interface showing the result of a executing a document clustering pipeline . . . . .	86
5.27	The user interface for predicting the the cluster model of a given design decision . . . . .	87
5.28	The high-level system design of the recommendation component . . . . .	88
5.29	A subset of concepts in the DBpedia ontology that are relevant for the recommendation component . . . . .	91
5.30	The system design of the ontology-based recommendation component . . . . .	95
5.31	Concepts and their relationships for persisting annotations and suggestions in the recommendation component . . . . .	95
5.32	The list of quality attributes and their subcategories in the ISO/IEC 25010 standard . . . . .	96
5.33	The high-level overview of the approach for recommending experts who could be involved in the ADM process . . . . .	98
5.34	An exemplary excerpt of an expertise matrix: rows in the matrix capture the expertise profiles of individuals and columns represent architectural elements . . . . .	100
5.35	The system design of the Akre-server component: the design shows the application controllers and their dependencies on the data models . . . . .	102
5.36	A datatable in ADeX shows the list of projects that have been imported from Jira . . . . .	104

---

5.37	User interfaces to import all the Jira issues of a project and to execute the processing pipeline for decision detection and annotation . . . . .	105
5.38	Quality attributes viewpoint: each individual bar represents a quality attribute and the segments within a bar indicate the design decision category . . . . .	106
5.39	Architectural elements viewpoint: each bubble represents an architectural element and the size of the bubble indicates the element's relevance to the project . . . . .	107
5.40	Expertise matrix viewpoint: rows correspond to individuals, columns capture architectural elements, and cells indicate the expertise of an individual on the corresponding architectural element . . . . .	108
5.41	A datatable showing the recommendation of experts and their expertise score for addressing open design concerns . . . . .	109
5.42	A datatable showing the list of automatically detected design decisions, their associated quality attributes and architectural elements . . . . .	109
5.43	Recommendations related to a design decision: annotated architectural elements, their alternatives, and similar design decisions made in the past . . . . .	110
6.1	The machine learning pipeline for design decision detection and classification; Classifiers: SVM, Naive Bayes, Decision tree, Logistic regression, One-vs-rest; n-grams: one to five; Split strategies: 90%, 80%, 70%, 60%, 50%; . . . . .	120
6.2	The influence of n-grams and split strategy on decision detection: increasing the training dataset increases the F-score and the variation of n in n-grams does not drastically affect the F-score . . . . .	122
6.3	The influence of n-grams and split strategy on decision classification: increasing the training dataset increases the F-score and the variation of n-grams does not have any noticeable affect on the F-score . . . . .	123
6.4	The evaluation results of the expert recommender for the Apache Spark dataset .	126
6.5	The evaluation results of the expert recommender for the Hadoop Common dataset	127
6.6	The evaluation results of the expert recommender for the Industry Project I dataset	128
6.7	The evaluation results of the expert recommender for the Industry Project II dataset	129
6.8	A Microsoft Word plugin for architectural recommendations: the plugin uses the recommendation services presented in this dissertation . . . . .	135
6.9	A chatbot for architectural recommendations . . . . .	136
7.1	The OODA Loop decision cycle; Adapted from [13] . . . . .	142
7.2	The BRM with the OODA Loop; BRM adapted from [14] . . . . .	143
7.3	The two-stage classification of cognitive biases using the OODA loop . . . . .	144
7.4	An example of a cognitive bias (planning fallacy) as documented in the bias catalog	145
7.5	A virtual meeting companion for supporting architects in online meetings . . . . .	147



---

## List of Tables

---

1.1	The description of concepts in the ADD model (taken from [3] and [4]) . . . . .	4
2.1	Requirements captured by Kruchten et al. [P1] for managing an Architectural Knowledge (AK) repository . . . . .	19
2.2	A taxonomy of design decisions: presented by Kruchten et al. in [P2] . . . . .	23
3.1	Architectural Knowledge Management (AKM) related use cases, their relevance to this dissertation, and their sources . . . . .	44
3.2	Architectural Decision Making (ADM) related use cases, their relevance to this dissertation, and their sources . . . . .	46
5.1	The list of quality attributes in the ISO/IEC 25010 standard and their keywords used for automatic rationale extraction . . . . .	97
6.1	Evaluation results of the automatic annotation of architectural elements in natural (English) language text . . . . .	112
6.2	Evaluation results of the recommendation of software and alternative solutions . . . . .	113
6.3	Exemplary recommendations for software solutions that are automatically generated by the recommendation component . . . . .	114
6.4	Exemplary recommendations for alternative solutions that are automatically generated by the recommendation component . . . . .	114
6.5	Rules for manual classification of design decisions into structural, behavioral, and non-existence/ban decision categories . . . . .	118
6.6	The configuration parameters used for training the machine learning classifiers . . . . .	120
6.7	Decision detection: confusion matrix of the SVM classifier with a linear kernel . . . . .	121
6.8	Decision classification: confusion matrix of the SVM classifier with a linear kernel . . . . .	123
6.9	The details about the dataset of four projects used for the evaluation of the expert recommendation system . . . . .	125



# CHAPTER 1

---

## Introduction

---

Software architecture is both an artifact as well as a process. A software architecture as an artifact can be thought of as a blueprint of a software system. This blueprint is created at different abstraction levels: high-level is typically referred to as software architecture and low-level or implementation-level, is considered as software design. The blueprint of a software system is generated based on a series of Architectural Design Decisions (ADDs) regularly taken by software architects and developers that have a far-reaching impact on the functional and non-functional properties of the resulting architecture and the corresponding software system.

Even though the term “architecting” is not defined in a standard English dictionary, the process of designing the software architecture is commonly referred to as “architecting” by some researchers and practitioners in the software architecture community. More commonly, the process is referred to as “architectural activity”, “design process”, or “software design”. This process of designing software systems is both a complex and a knowledge-intensive activity [15]. The knowledge-intensive activity includes making ADDs about the structural, behavioral, and organizational aspects for conceptualizing and realizing the blueprint of the corresponding software system. Since, software architecture as an artifact is created at different abstraction levels, ADDs too are taken at different abstraction levels such as high-level, medium-level (detailed design), and realization-level (implementation or code specific) [16]. The decisions at different abstraction levels are related to each other and form a tree structure (referred to as the funnel of decision-making [17]). Moreover, the decisions at a higher level of abstraction constraint or influence the decisions at lower levels. The high-level decisions for instance include the selection of architectural styles, medium-level decisions correspond to adding, updating, or removing software components, interfaces, third-party software modules from the architecture, and realization-level decisions include changes made to the source code of software systems. Such decisions taken by architects and developers, in turn, result in the architecture of the software system. Hence, over the last decade, with the representation of Architectural Design Decision (ADD) as a first-class

entity [18–20], there has been a paradigm shift in how we view software architecture. Today, software architecture is considered as a set of architectural design and ADDs [3,21].

$$\text{Software architecture} = \text{Software design} + \text{ADDs}$$

This new perspective of representing software architecture as (a) the result of ADDs taken by architects and developers through an Architectural Decision Making (ADM) process and (b) focusing on ADM process that results in the software architecture can be considered as one of the key advancements in software architecture research. This is because, ADDs represent the rationale that motivates the selection of architectural elements within the software architecture [22]. The need for documenting ADDs and their associated concepts including architectural concerns, alternative architectural solutions, and rationale for ADDs has been emphasized both in research as well as in industry [23]. Furthermore, as discussed in Chapter 2, from the year 2005 to 2011, there have been a plethora of Architectural Knowledge Management (AKM) tools presented at research conferences that provide stakeholders the capability to document and manage ADDs and their related concepts. Unfortunately, even though, architects understand the benefits of explicitly documenting ADDs [24], in practice, those decisions are rarely captured [25] in those AKM tools, and they are lost over time. However, we argue that even though ADDs are not explicitly documented, those ADDs are implicitly captured in various systems which are regularly used by architects and developers in their day-to-day activities. These systems, for instance, include Wikis, Issue Management Systems (IMs), versioning systems, and chat clients. By applying the approaches presented in this dissertation, ADDs implicitly captured in those systems can be automatically extracted and analyzed by stakeholders in software projects.

This dissertation focuses on supporting architects and developers during the ADM process in large software projects. We present a series of approaches that help to address specific use cases related to the ADM process. The use cases include the following:

- (a) extraction and classification of design decisions from disparate information sources;
- (b) annotation of architectural elements within textual information;
- (c) recommendation of alternative decision options;
- (d) reasoning about design decisions;
- (e) identification of similar decisions made in the past;
- (f) recommendation of experts for addressing new design concerns;

These use cases are stitched together coherently, and the approaches for each of the use cases are realized by independent components which are integrated within a component-based framework with adequate tool support. We refer to this system as “ADeX” (**A**melie - **D**ecision **eX**plorer). Over the past four years, ADeX has been implemented based on the findings from the existing AKM tools, the future research directions proposed in the literature, and collaboration with our industry partner (an architecture definition and management department within a large multinational company). Furthermore, each component has been evaluated through quantitative analysis and the applicability of ADeX for supporting architects and developers during ADM is demonstrated through qualitative studies.



## 1.1 Problem statement and research questions

Since the explicit representation of ADDs as first-class entities in 2004, many researchers have proposed meta-models for managing ADDs [1, 2]. Subsequently, based on those models many AKM tools [26–28] have been implemented to support the documentation of ADDs and its associated concepts. As shown in Figure 1.1, the concepts within the ADD model include the rationale behind a decision, concerns addressed by a decision, and the architectural elements affected by a decision.

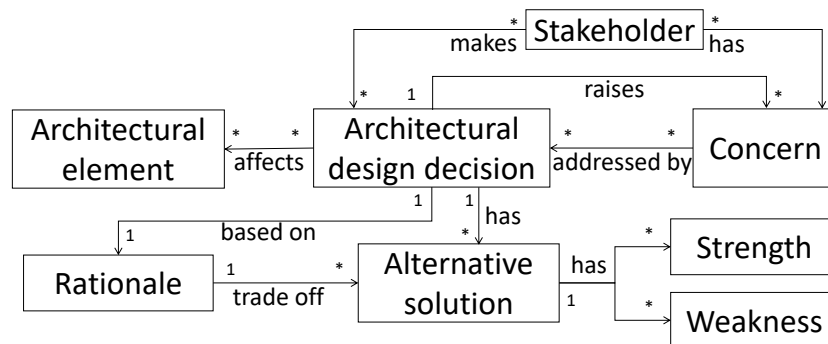


Figure 1.1: The representation of ADD and its associated concepts in an ADD model. The concepts in the ADD model have been derived from [1–4] (cf. Section 2.2.2.1 for a detailed discussion)

We have consolidated the core concepts in the ADD model shown in Figure 1.1 based on the literature review of forty-two publications presented in Section 2.2.2.1. In this dissertation, we use the definitions of the concepts in the ADD model as presented in Table 1.1. Note: these definitions are primarily taken from [3] and [4].

As shown in Figure 1.1, **stakeholders** have (quality) **concerns** which are addressed by one or more **architectural design decisions**. The decisions taken by stakeholders affect one or more **architectural elements** and may raise new concerns. An ADD is [should be] made by considering multiple **alternative solutions**. The selection or the rejection of alternatives based on their **strengths** and **weaknesses** forms the **rationale** of a design decision. Documenting ADDs helps stakeholders to (a) avoid architectural knowledge vaporization in large software projects, (b) understand and reason about a software architecture during both the development and maintenance phases [29], and (c) reuse knowledge about already made decisions while making new decisions in similar context.

Furthermore, practitioners are emphasizing the need for capturing and reusing ADDs to avoid knowledge vaporization and to reduce the time and effort involved in addressing recurring design problems. To enable the documentation process, industry-standard software architecture templates (for example, arch42<sup>1</sup>) also provide placeholders to capture ADDs. However, the manual effort [30, 31], time, and cost [24] involved in the documentation process are a concern for stake-

<sup>1</sup><https://arc42.org/>

Concept	Description
<b>Stakeholder</b>	Stakeholders are individuals, groups, or organizations holding <i>concerns</i> for the <i>system</i> of interest. Examples of stakeholders: client, owner, user, consumer, architect, developer, and auditor.
<b>Concern</b>	A concern is any interest in the system. Examples of concerns: (system) purpose, functionality, structure, behavior, cost, non-functional requirements.
<b>Architectural element</b>	Any item in an architecture description is considered an element. Examples of architectural elements: views, styles, patterns, components, modules, technologies.
<b>Architectural design decision</b>	A description of the set of architectural <i>additions</i> , <i>subtractions</i> , and <i>modifications</i> to the architecture, the rationale, and the design rules, design constraints and additional requirements that realize one or more concerns on a given architecture.
<b>Rationale</b>	Architectural rationale records the <i>explanation</i> , <i>justification</i> , or <i>reasoning</i> about architectural decisions that have been made and alternative solutions not chosen.
<b>Alternative solution</b>	Alternatives are potential solutions to the requirement the design decision addresses. The <i>choice</i> is the decision part of an architectural design decision; it selects one of the considered alternatives.
<b>Strength and Weakness</b>	The <i>pros</i> and <i>cons</i> of each alternative provides architects the basis for a choice and forms the justification for that decision.

Table 1.1: The description of concepts in the ADD model (taken from [3] and [4])

holders, and its immediate benefit is not visible [32]. Stakeholders also consider capturing ADDs to be intrusive and boring [24].

Moreover, with the rapid adoption of agile methodologies for software development, ADDs both in large Open Source Software (OSS) and in industrial projects are scarcely documented [33]. However, stakeholders involved in projects that follow this agile movement, tend to use agile project management tools such as issue trackers and version control systems [34,35]. In such projects, even though ADDs are not explicitly documented, they are implicitly captured in different systems including project management, issue management, source code version management, and meeting recording systems [36,37]. In this dissertation, we present ADeX which follows a bottom-up approach to AKM with the focus on automating the AK curation process. ADeX helps to automatically extract, annotate, and generate specific views on AK to support ADM process. Here, the bottom-up approach refers to the idea that ADeX extracts information and derives AK from project data without the need for stakeholders to explicitly and manually capture ADDs in a specific AKM tool and in a particular format (in a top-down manner) to receive recommendation support.

Finally, realizing the fact that software design is a complex decision-making process has engaged researchers in borrowing and incorporating ideas related to how people make decisions from philosophy, cognitive science, and sociology. As indicated in [38–40], research in the area of

understanding and supporting architects during the ADM process is still in its infancy. There have been some efforts in understanding the ADM process in software architecture (“How ADDs are made?”) which includes addressing research questions such as “What is an ADD?”, “Why is an ADD made?”, “When is a decision taken?”, and “Who makes ADDs?” [38,41]. Furthermore, another characteristic of decision making is choosing the “best/suitable” solution from a set of choices. The selection process raises questions such as “What are the alternative solutions?”. In this dissertation, once we extract ADDs from project’s textual information, we address the aforementioned questions by providing tool support that could be integrated within software architects’ and developers’ working environment.

In particular, we address the following research questions:

- RQ 1:** How to consolidate project data from disparate information systems?
- RQ 2:** How to extract design decisions from textual information?
- RQ 3:** How to identify architectural elements and recommend alternatives to support ADM?
- RQ 4:** How to identify the rationale behind a design decision?
- RQ 5:** How to identify similar design decisions made in the past?
- RQ 6:** Who should be involved in making a design decision?

The first two research questions are related to extracting project information from different data sources (e.g., Jira tickets, architectural documents, etc.) and then, automatically identifying the already made decisions from those sources. In this dissertation, we only focus on the information extracted from the data sources that are in natural (English) language text. Hence, RQ 2 deals with identifying design decisions in textual information. The remaining research questions focus on supporting architects and developers during the ADM process. The research question - RQ 3 aims towards annotating the identified ADDs with information about the associated architectural elements and suggesting alternative options that can be considered during the ADM process.

Since quality attributes of a system are the drivers of ADDs and form the rationale behind a decision, RQ 4 investigates the challenge of identifying the rationale behind an ADD. Readers should note that RQ 4 has already been explored by many researchers including Burge, Dutoit, Tang, and Paech in the context of rationale-based software engineering [42–45]. Some of the above works have been discussed in the literature review section presented in Chapter 2. Furthermore, as discussed in Chapter 5, we address RQ 4 given its relevance to ADDs and for the completeness of this dissertation using a keyword-based approach derived from the works of [46–48].

RQ 5 emphasizes identifying similar decisions made in the past so that architects and developers can learn from previous experiences while making new ADDs. Finally, given that ADDs are made in groups, addressing RQ 6 helps to identify experts in specific architectural topics who can be involved in making new ADDs.

## 1.2 Contributions of this dissertation

In this research endeavor, we aim to address the aforementioned research questions related to ADDs and ADM by providing tool support that could be integrated within software architects' and developers' working environment to:

- (a) Extract and reuse AK related to similar ADDs made in large software projects,
- (b) Recommend alternative architectural solutions that could be considered during ADM,
- (c) Highlight and reason about the rationale behind ADDs, and
- (d) Recommend experts to address specific ADDs.

In the subsequent chapters, we elaborately discuss how these features are addressed within the proposed conceptual framework, and the corresponding implementation named ADeX. To briefly summarize, ADeX serves the following purposes. Firstly, ADeX is used for getting insights, especially into large software-intensive projects. Architects and developers can use ADeX to browse through projects and get a quick overview of the already made design decisions. These design decisions are automatically identified, for instance, from issues in an Issue Management System (IMS). Further information regarding those design decisions can be explored; such as, the affected quality attributes and architectural elements (concepts like databases or concrete technologies like SQL). Project managers could also use the overview of the concepts and technologies used in a project for staffing purposes. Secondly, ADeX is used to support architects and developers during the decision-making process. For instance, unresolved design concerns from an IMS are extracted into ADeX's knowledge base, and after preprocessing, architects get recommendations for resolving those concerns. ADeX helps stakeholders answer questions such as "who should be involved in ADM?" and "which similar ADDs have been made in the past?".

Furthermore, architects instead of exploring the solution space, often choose solutions (for example, software products) based on their intuition and past experiences. We, therefore, integrated a recommendation mechanism that shows software solutions and alternatives related to architectural elements in a given design decision. The recommendations trigger the mental thought process of architects and help them to reflect on their choices. The components within ADeX that provide these features have been publicly made available on Github<sup>2</sup>. Moreover, the datasets that were created to train the Machine Learning (ML) models to automatically extract decisions and to find similar decisions made in the past have also been made public. This contribution will serve as a starting point for researchers to further reference and investigate the design decision detection and classification problems.

The use cases tackled by ADeX have been derived from the future research directions shared by researchers in the software architecture research community as well as from practitioners in an industry. The models and approaches used in ADeX builds upon the existing research work; for instance, the recovery of design decisions relies on the definition of concepts in Kruchten's AK ontology [18]; the dynamic AK model of ADeX's knowledge is derived from [11]; the rationale extraction uses the quality keywords documented in [48]; and the expertise matrix used to identify individuals for decision making is based on the expertise browser presented in [49].

---

<sup>2</sup><https://github.com/sebischair>

Our research on automatic curation of ADDs complements the current research activities in the direction of the application of artificial intelligence for AKM. For instance, the recent research initiatives such as Continuous Usage- and Rationale-based Evolution Decision Support (CURES)<sup>3</sup> can use Amelie - Decision eXplorer (ADeX) and our lessons learned as a building block for supporting developers during software evolution. The authors of [50] and [51] refer to design decisions and the related rationale as design knowledge and argue that design knowledge is captured informally in Wiki, issue tracking, version control, and chat systems using natural language text. To overcome the challenge of intrusiveness of the existing AKM tools, Kleebaum et al. [52] present a plug-in for Jira named Jira DecDoc that integrates into developers' working environment to support the capture and use of design knowledge. Jira DecDoc uses the decision documentation model [53,54] that supports developers to incrementally capture design knowledge in their sprint cycles. The users' needs form the usage knowledge about the system and is reflected by features and tasks in Jira. While implementing each feature in a feature branch, developers can document the design alternatives, their strengths and weaknesses, and the choice (decision) taken to implement the corresponding feature. We believe that our approach that uses supervised ML to automatically identify design decisions from issues captured in Jira as well as the work of Alkadhi et al. [55,56] that extracts rationale in chat messages can be used to prepopulate the design knowledge model in legacy systems for retrospective analysis.

One of the drawbacks of the application of supervised ML techniques for automatic retrieval of design knowledge from natural language text is that it requires substantial effort to prepare the training dataset and to tune and train the ML models. Moreover, the generalization possibilities and transfer learning of models across projects in different domains are yet in the scope of future investigation. We agree with these critiques that are also mentioned in [57,58] and we have discussed these challenges in our lessons learned chapter.

It is also vital to inform the readers of this dissertation the assumptions and constraints that apply to the presented approaches and the implemented system (ADeX). Since a primary focus of our work lies in providing automation support through components that learn from ADDs made in software projects, we emphasize on medium- and realization-level decisions (cf. Section 1 - ADDs are made at different abstraction levels). These ADDs are less likely to be influenced by projects' business and political aspects and are typically and implicitly captured in disparate agile project management systems. Furthermore, we make the following assumptions based on the past empirical studies conducted by researchers working in the area of AKM:

- Architects do not take a rationalistic approach but favor a naturalistic ADM [39,41].
- Software architects make ADDs based on their expertise and past experiences [39,59].
- The current state of the project and past project artifacts are observable given a predefined domain model [27].

The contribution of this dissertation is a set of approaches integrated in ADeX that supports (junior and newly recruited) software architects and developers in industry:

- First, to get accustomed to the assigned long-running software projects by exploring and understanding design decisions that were already made to address specific quality concerns.

---

<sup>3</sup><https://se.ifi.uni-heidelberg.de/research/projects/cures.html>

- Second, to learn from decisions that were made by other architects in similar projects.
- Third, to trigger the thought process through the recommendation of alternatives so that they consider different options before making a design decision.
- Finally, to encourage the culture of explicitly documenting or even labeling design decisions by highlighting its benefits with regards to traceability and impact analysis during the development and maintenance of software systems.

### 1.3 Outline of this dissertation

As shown in Figure 1.2, this dissertation is organized into seven chapters. Chapter 1 provides an introduction to ADDs and ADM and motivates the readers about the new perspective of viewing software architecture as a set of ADDs. The contributions of this dissertation, as well as the research questions addressed in this dissertation, are also discussed here.

In Chapter 2, the state of the art analysis of topics concerning ADM is presented. Following a semi-systematic literature review process 227 publications belonging to different genres including ADDs, (group) decision making, AKM, ADD models, and tools are presented in depth.

The analysis results from the literature review along with the inputs from our industry partner form the use cases for the conceptual framework and the corresponding ADeX system presented in this dissertation. These use cases are consolidated in Chapter 3.

Chapter 4 and Chapter 5 describe in detail the core contributions of this dissertation. The conceptual framework and the implementation of the ADeX system based on the framework are covered in these two chapters. The design of each of the components and the algorithms used within ADeX are elaborated in Chapter 5.

The results of both the quantitative and qualitative evaluation of the components in ADeX are discussed in Chapter 6.

Over the last four years, we have received many suggestions, criticisms, and improvement areas both from the industry as well as from the software architecture research community. We share our lessons learned and conclude with an outlook in Chapter 7.

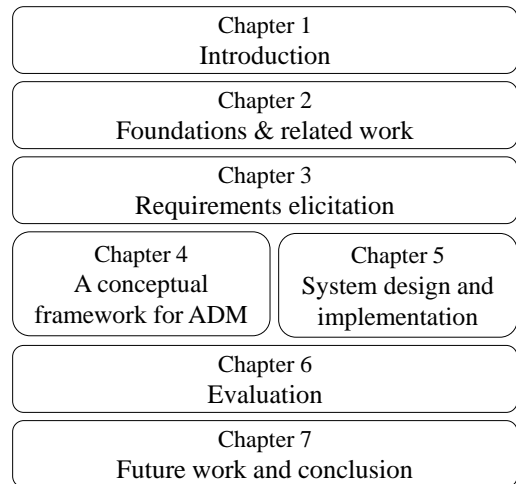


Figure 1.2: Outline of this dissertation

**Note:** Some of the sections in this dissertation have been taken from the publications [27,60–64] wherein, the author of this dissertation is also the first author of those publications.

This chapter provides readers of this dissertation the necessary background and the state-of-the-art analysis of the topics concerning ADM in software engineering. It covers various aspects with regards to ADDs, such as models of AK with the focus on design decisions, tool support for ADM, ADM techniques, and factors influencing ADM.

The research gaps, challenges, and requirements identified by researchers working in the area of ADM are highlighted (as italics in the text) through a semi-systematic literature review. Those highlights presented in this chapter form the foundations, in the form of use cases, of the ADeX system presented in this dissertation.

## 2.1 Research methodology for conducting the semi-systematic literature review

Various researchers have proposed guidelines and protocols for conducting systematic literature reviews in software engineering (cf. [65–68]). We use the guidelines prescribed by Kitchenham and Charters in a technical report [5]. As discussed by Kitchenham and Charters, there can be many reasons for undertaking a systematic literature review. We conducted a systematic literature review to provide the necessary background to position our work on ADM. This justification for performing the review is part of the “*planning the review*” stage within the systematic review process.

In essence, as shown in Figure 2.1, the review is broken down into three stages, namely, *planning*, *conducting*, and *reporting* the review. In the planning stage, the need for the review (as discussed before), the research questions related to the study (cf. Section 1.1) are presented. Then, the literature review protocol is presented. In the second stage, as per the review protocol, the

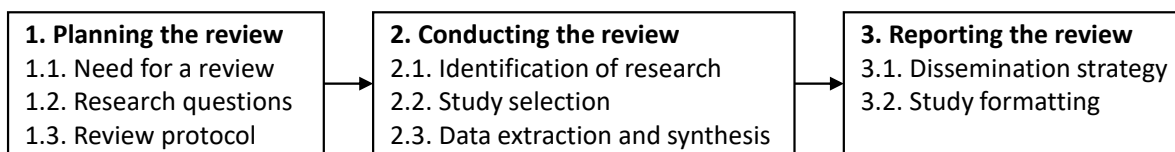


Figure 2.1: Planning, conducting, and reporting a systematic literature review: as prescribed by Kitchenham and Charters [5]

actual review is performed. And finally, the review is presented, in our case, as part of the documentation within this dissertation.

It should be noted that we did not conduct specific steps within the review stages which are also not considered mandatory by Kitchenham and Charters [5]. Those steps include commissioning the review, evaluating the review protocol, and evaluating the review report.

As part of the review protocol that was followed while conducting the review, the following strategies were used:

- For the identification of research, we defined the inclusion and exclusion criteria
- For the selection of relevant publications, we described the search strategy
- Data extraction was performed based on the inclusion and exclusion criteria
- Finally, the collected references were analyzed and summarized

Each of these steps are elaborated in detail in the subsequent subsections.

### 2.1.1 Inclusion and exclusion criteria for identifying research publications

For the inclusion criteria as part of the selection process, we considered those publications that:

- + refer to software architecture or software design
- + discuss ADDs as the main topic (this could include AKM, ADD models, and ADM)

Since the search query (as discussed in the next subsection) was formulated using English terms, we restricted ourselves to those studies published in the English language. Furthermore, to retrieve all relevant studies over the years, no lower boundary was set on the publication date. However, since the search queries were executed on 17.07.2018, that is considered as the upper boundary of the publication date. The following list of exclusion criteria were considered for the literature review:

- Publications which are not in the English language.
- Studies published after 17.07.2018.
- Studies published by the author of this dissertation. Those publications are discussed in detail in this dissertation.



- Publications describing a conference, a workshop, or a keynote talk.
- Publications that could not be accessed (even behind our library's proxy - four in total).
- Publications that discuss the architecture of a specific system (for example, architecture of a decision support system). Such papers were retrieved due to the search query construct and could not have been avoided.
- Publications that focused on very specific specialized related topics, such as requirements negotiation among stakeholders, model-driven approaches for realizing design decisions, modeling service-oriented process decisions, and pattern-based approaches.

### 2.1.2 Search strategy for identifying relevant research publications

The following four digital libraries were chosen for retrieving the key publications. Since we were interested in ADDs, we considered these four important publishers and indexes that contain software architecture publications:

- ACM Digital Library
- IEEE Xplore Digital Library
- ScienceDirect
- Springer Link

These four publishers publish proceedings of the major software architecture conferences and workshops including, (a) International Conference on Software Engineering (ICSE); (b) Working IEEE/IFIP Conference on Software Architecture (WCSA); (c) European Conference on Software Architecture (ECSA); (d) International Conference on Software Architecture (ICSA); (e) Working on Sharing and Reusing Architectural Knowledge (SHARK); (f) Workshop on Decision Making in Software ARCHitecture (MARCH); and (g) International Conference on Software Architecture Workshops (ICSAW).

It can be argued that the above list is not exhaustive; however, the sources mentioned above cover the major interest groups of researchers working in the area of software architecture and those who publish in "A-ranked" conferences and workshops. Hence, the retrieved publications would include relevant publications representing the state-of-the-art in the area of ADM.

To ensure all relevant publications were retrieved, we formulated a broad search query based on the inclusion criteria. First, we wanted to include publications in the area of "software architecture" or "software design" (Q1). Second, in the context of Q1, we considered those publications that discuss "decision making" or mentioned "design decisions" (Q2). Finally, given that both Q1 and Q2 had to be satisfied, they were combined using an AND clause.

Q1: ("software architecture" *OR* "software design")

Q2: ("decision making" *OR* "decision-making" *OR* "design decisions")

FSQ: (Q1 *AND* Q2)

The final search query (FSQ) implies that at least one term from both Q1 and Q2 must appear at least once. The FSQ was matched against the *title*, *abstract*, and *keywords* of the publications in each of the digital library. In each of the above digital libraries, the search was executed using the advanced search option that provided the inclusion of such joined queries. It should be noted that only for Springer Link, we could not match the query against the three publication fields (title, abstract, and keywords). Instead, we had to perform a full-text search on the publications that resulted in a larger number (1,455 publications) of retrieved publications as compared to the results of other digital libraries.

### 2.1.3 Data extraction and synthesis for conducting the literature review

On executing the search query on individual digital libraries on 17.07.2018, we found 90 publications in ACM Digital Library, 141 in ScienceDirect, 1,455 and 230 publications in Springer Link and IEEE Xplore Digital Library respectively. These search results were exported to a CSV file for further analyses. Next, as shown in Figure 2.2, each of the publication was filtered by reading the title and abstract according to the aforementioned inclusion criteria. This filtering step resulted in 69, 26, 60, and 163 publications from ACM Digital Library, ScienceDirect, Springer Link, and IEEE Xplore Digital Library respectively. These 318 publications were merged into a new CSV file. Out of these 318 publications, we found 17 publications that occurred more than once and were removed. For each of the remaining 301 publication, their full text was thoroughly read. During this step, many publications were removed as per the already mentioned exclusion

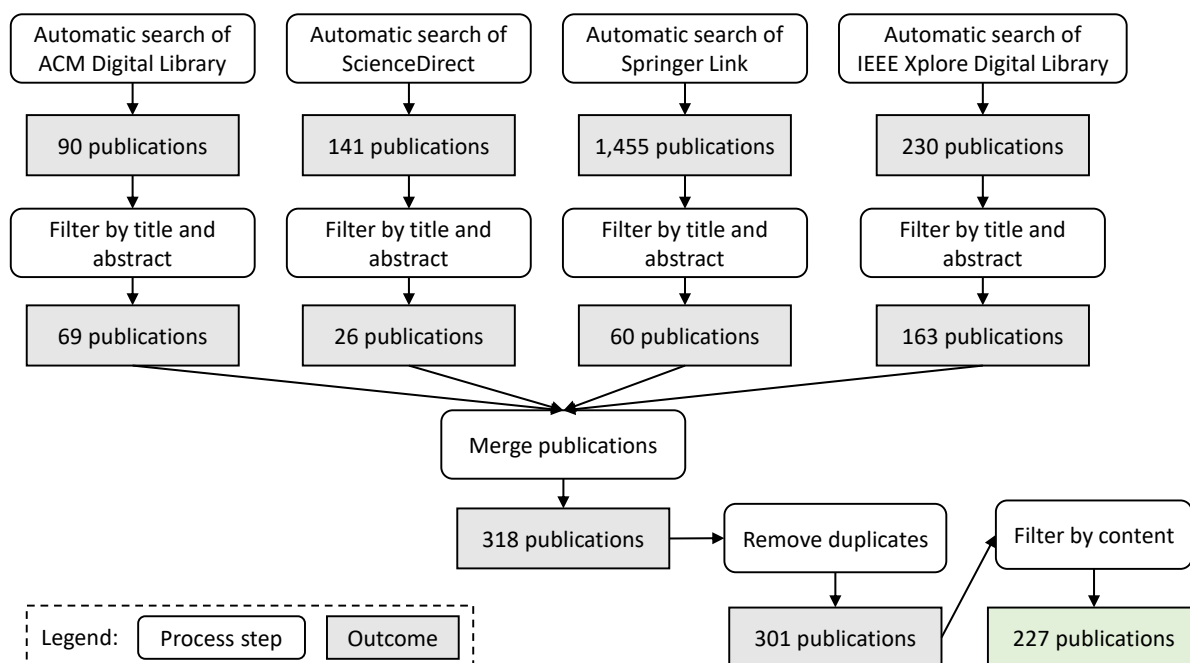


Figure 2.2: The search strategy that resulted in 227 publications for conducting the semi-systematic literature review

criteria. This final step resulted in a total of 227 publications which were summarized and the findings based on those 227 publications are presented in this dissertation.

## 2.2 Review results of the identified 227 publications

Figure 2.3 shows the distribution of 227 publications (according to their publication year) that were selected for the analysis. The first publication about design decision was presented in 1980. Even though we had an upper bound for the publication date (the date when the search query was executed - 17.07.2018), we found three papers that were yet to be published in 2018 and were already available as pre-prints. Those three papers were also included in our study.

Note that in Figure 2.3, we see a steep increase in the number of publications per year related to design decisions after 2004 (referred to as a paradigm shift in software architecture). As also indicated by Capilla et al. [1], the original paradigm was purely technical and viewed architecture with components and connectors, views, patterns, reference architectures, etc. The second or the new paradigm looks at architecture from a socio-technical standpoint. Since 2004, researchers in software architecture community started to look at architecture not only as a composition of components and their relationships but instead began to reflect on how those components and relations come into existence (how architects reason and make decisions).

This trend of viewing software architecture as a set of design decisions can be attributed to the works of authors including Jansen, Bosch, and Kruchten. Since 2004, there has been a plethora of publications addressing various aspects of design decisions, for example, models and tools for managing design decisions, different decision-making strategies, and factors influencing the decision-making process. Today, in 2018, we observe at least one research track dedicated to architectural design decisions and decision making in software architecture conferences such as ECSA and ICSA.

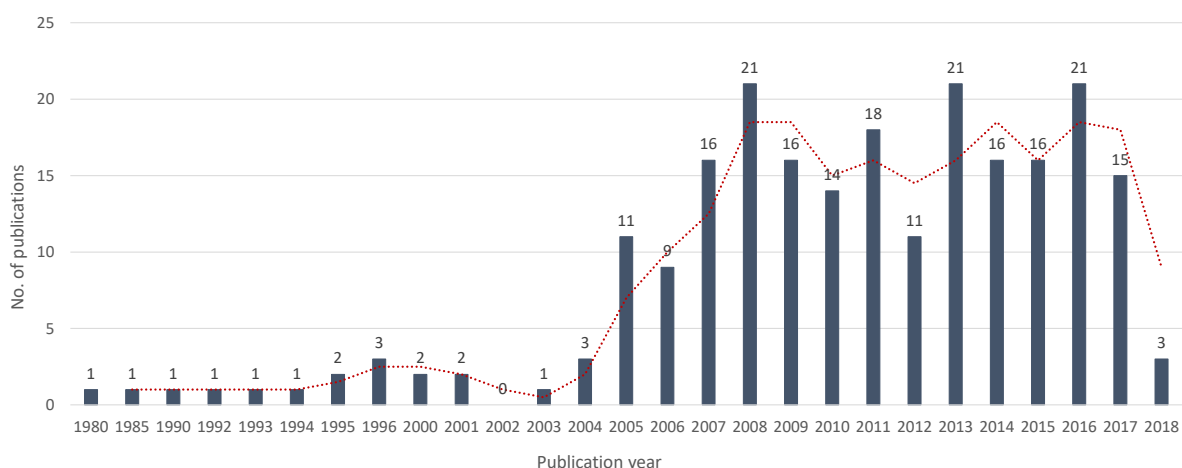


Figure 2.3: The trend in the number of publications related to AKM, ADDs, and ADM distributed over the past years

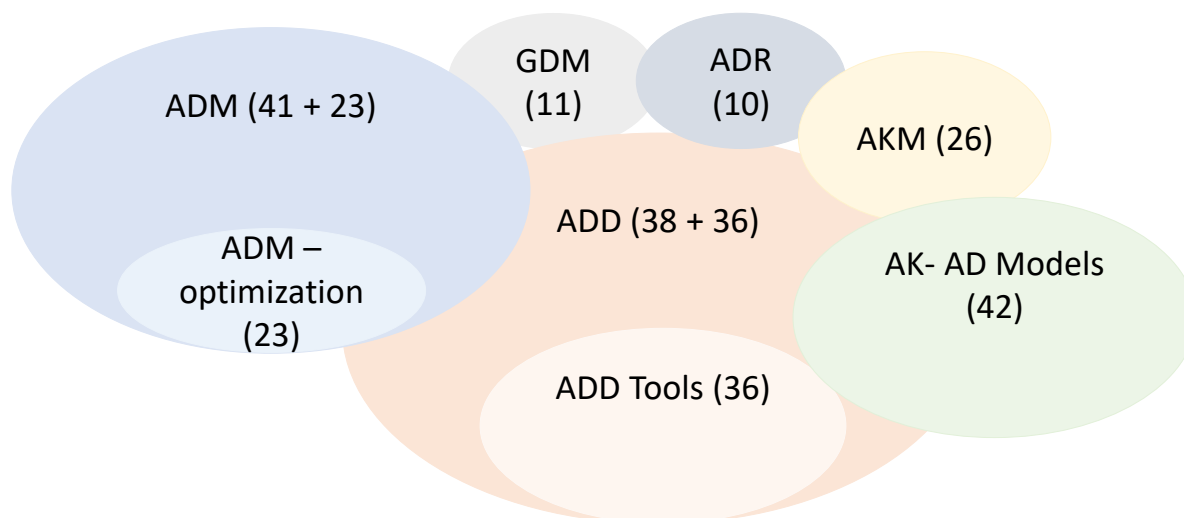


Figure 2.4: The classification of the identified 227 publications into different AK categories

To analyze the 227 identified publications, we categorized them into different categories, namely, AKM, ADDs, AK/AD models, Architectural Design Rationale (ADR), ADM, and Group Decision Making (GDM). Many of the publications cover more than one topic. While pointing out this overlap, we discuss them only in one of those topics. Furthermore, one could argue that GDM is part of ADM or managing ADDs is part of AKM. It should be noted that the publications were categorized into the aforementioned topics only for the convenience of summarizing the results. Depending on the context and the theme of the publication, we categorized those publications into one of those topics. For instance, if a publication specifically emphasized GDM, even though presented, for example, an ADM strategy, it was sorted into the GDM category.

Accordingly, Figure 2.4, shows the number of publications per topic. For instance, we have a total of 74 publications listed under ADDs; out of which, we have 36 publications that provide tool support for managing ADDs. Twenty-six publications focus specifically on managing AK and ten publications discuss the topic of ADR. We also categorized twenty-three publications within ADM that model ADM as a multi-objective problem.

In the following sub-sections, we summarize the publications starting with the broad area of AKM, followed by ADDs, AK models, and ADR. In the end, we present those publications related to the topic of ADM and GDM. In each of the following sections, we have tried to discuss the publications in a chronological order of the publication date. However, in some situations, we have also grouped some publications to fit the context.

### 2.2.1 Architectural knowledge management

One of the very early works by Parnas and Clements [P3] emphasizes that designing software is quite an irrational process. They state that the rationale behind the long sequence of design decisions made by developers are rarely explained. Furthermore, they also argue that software design cannot be entirely rational for many reasons, such as requirements and domain understanding

evolve and humans cannot comprehend all the information for rational thinking. However, they indicate that developers try to capture the rationale after the fact. They suggest that by faking a rational design process (i.e., by following an ideal process as closely as possible), developers can get guidance on how to proceed. Finally in this paper, the authors discuss the *need for documenting design decisions*, that is, recording all design alternatives that were considered and the rationale as to why an option was considered or rejected.

In [P4], the authors conducted multiple interviews and were the first (in our review list) to observe that design decisions are lost (forgotten) from one design meeting to the next. They indicate that previously made decisions are commonly questioned in the subsequent ones. To improve the design process, they suggest the *need for integrating, documenting, and sharing AK*. [P5] also states that architectural degradation during software development happens when already made design decisions are modified or when they are not recorded and forgotten over time. Another publication [P6] also highlights that AK (specifically ADDs) reside in architects' mind and are lost over time. Explicitly modeling and capturing design decisions helps to manage the rationale about the elements within a software architecture. [P7] presents the interview results of 279 architects and highlights that architects spend more time on making ADDs and less time on documenting those decisions.

Farenhorst et al., in [P8,P9], present the first generation AK sharing tool named EAGLE with integrated features to support the ADM process. This tool combines codification techniques (design decision repositories, document management facilities) with professionalization mechanisms (yellow pages, discussion forums) to *deliver AK to the "right" people at the "right" time*. Furthermore, they also share observations from industry about AK sharing. They compare and discuss six different software architecture tools. Moreover, the same authors based on four case studies [P10] indicate that architects would use any AK tool support, if and only if, *AK sharing is versatile and lightweight*. The tools should explicitly support *managing decisions, searching relevant information*, and even *finding the right colleagues*. [P11] also compares different approaches based on knowledge modeling, decision-making, and rationale management capabilities. The findings in [P11] indicate that there is a limited support for identifying Architecturally Significant Requirements (ASRs) and for managing cross-cutting ADDs.

For sharing high-quality AK across projects in different domains, Liang et al. [P12] present an indirect mapping approach which compares the semantic distance between different AK models to quantify the AK sharing quality and to find a high-quality AK model with the shortest semantic distance. On a related note on sharing AK, in [P13], Van Vliet discusses a research project named GRIFFIN that aims to address the challenges of capturing AK from a decision-oriented perspective. The author argues that along with the design, the knowledge that has led to that design is also essential, leading to the point that the so-called *AK concerns the set of design decisions and their rationale*. Furthermore, the author also highlights that design decisions address one or more concerns and there may be multiple ways of resolving them. Hence, a decision is a choice among alternatives that meet some favorable characteristics (rationale) and impacts the subsequent decisions. De Boer and Farenhorst [P14] after a systematic literature review on the definitions of AK noticed that most of those definitions focus on design decisions. De Boer also called for "closer cooperation between architecture and requirements engineering communities" since there seems to be "no fundamental differences between architecturally significant

decisions and ASRs” [P15]. Chen in [P16] also proposes the co-development of requirements and architecture by using the (implicit) links between ASRs and ADDs.

A tool named Knowledge Architect uses a domain-specific AK model to annotate architectural documents and stores those annotations in a knowledge base [P17]. The semantically enriched knowledge instances are then used to *facilitate AK retrieval* and to improve the understandability of architectural documents. However, it should be noted that the task of creating knowledge elements in the AK model is a manual and an effort intensive activity.

A plugin for Sparx Enterprise Architect called ADMentor is presented in [P18]. This tool maintains decisions in a backlog and allows tagging them as, for instance, urgent for a design iteration. It lists the decisions in a table and also facilitates filtering and ordering features. ADMentor was validated by capturing design decisions in 85 cloud applications and 75 workflows.

Another tool named ArchiMedes built on top of MediaWiki (a semantic KM platform) is presented in [P19]. The tool’s core capabilities include *publication, enrichment, analysis, and integration of AK from ArchiMate-compliant repositories*. In our approach, presented in this dissertation, those features are provided by the SocioCortex platform which is also based on a HybridWiki meta-model which is similar to the semantic MediaWiki platform’s model.

Tang et al. [P20] present a comprehensive comparison of five AKM tools. Based on the comparison, they stated the need for improved *support for AK sharing, collaborative work, and personalization mechanisms*. Furthermore, in this publication, they also define four categories of AK, namely, context knowledge, general knowledge, reasoning knowledge, and design knowledge. These categories are discussed in Chapter 5, as they were used to build the knowledge base of the system presented in this dissertation.

A systematic mapping study of fifty-five publications is presented in [P21]. In this study, the authors analyzed which AK-based approaches were used in which architectural activities. They found that even though AK capture and representation is popular, *AK recovery* (for instance, by documenting past design decisions) is seldom used in software architecture. Another literature review in 2014 analyzed the models of AKM approaches [P22]. The authors admitted that their review results did not show any strong evidence for the support of core AKM activities (e.g., AK maintenance) and they found only weak evidence for AK sharing and reuse.

To enable an integrated AK extraction and sharing process, the authors of [P23] present a method that uses the trust of stakeholders on AK evaluation. They emphasize collecting and using valid AK and promoting cooperation among individuals in different architectural activities. Furthermore, cooperation among individuals should not be limited to architects and developers. [P24] calls for improved collaboration between requirements engineers and architects. It is suggested that architects might benefit from *explicitly linking requirements and decision elements*.

Soliman et al. [P25] discuss a significant challenge with the existing AKM tools, which is, manually capturing and maintaining the AK (e.g., architectural solutions and design decision rules) in AK repositories for supporting architects. Creating and maintaining such AK repositories is time-consuming and a tedious process. The amount of AK gathered manually is somewhat incomplete thereof. Hence, authors call for *efficient methods for capturing and maintaining AK*. We agree with Soliman et al. and the bottom-up approach for AKM presented in this disserta-

tion correlates with that idea. In [P25], authors specifically focus on technology decisions. They classified StackOverflow posts into different types of architecture- and technology-related knowledge (pure programming posts, architecture-relevant posts, and cross-architecture/programming posts). Such classification can be used to train ML models to structure future posts in their respective category automatically. As a follow-up, to support architects while solving architectural problems, the same authors developed an ontology (with 3,800 AK concepts) by analyzing 65 architecturally-significant StackOverflow posts [P26]. On the related note, the authors of [P27] also highlighted that eventually, ADM is constrained by technology decisions.

*AK integration* is not extensively supported by many of the existing AKM tools. The authors of [P28] highlight an important point: “architects and developers do not always document and share AK in organization-internal systems, but for the sake of convenience just refer to a diverse collection of organization-external information sources (e.g., blog posts, Q&A portals, GitHub repositories).” While the authors discuss the need to integrate AK from different information systems and mention a semantic model to consolidate, synthesize, and disseminate AK, they do not elaborate on the concepts within their semantic model.

One of the most relevant and motivational publications relevant for this dissertation is by Capilla et al. in 2016 [P29]. The lack of widespread adoption of AKM approaches even with the improvements over the years, led Capilla et al. to investigate the success and shortcomings of current AKM approaches and to document what industry needs from AK. In this work, the authors touch upon various aspects including the challenges in the current AKM practices on how architects and developers make decisions. These challenges are listed below:

- A uniform AKM approach cannot cover all AK variability. Each of the AKM approaches has its own set of concepts and relationships in their AKM models. Hence, *solutions should cater to the specific needs of practitioners.*
- The one-size-fits-all approach does not work: *tools and AK models should be organization-, domain-, or project-specific. Users should be able to configure them accordingly.*
- Existing AKM tools do not integrate with other commercial tools. There is a need for a *highly integrated, easy-to-use, and shared AKM tool.*
- *AKM tools should interoperate with tools from different SDLC phases to establish traceability* (between requirements, decision decisions, architectural elements, and code). Creating traces can help estimate the impact of a decision on other artifacts.
- For an AKM tool to be adopted, *it must be lightweight.*
- *AKM tools must be descriptive and not prescriptive* (give advice and not make decisions).
- *AKM tools should substantially improve their usability and their visualization features.*
- *AKM tools should (semi-) automate some of the AKM use cases.*
- *AKM tools should be able to track evolving and runtime decisions.*
- *Architects base their decisions on their own experiences and expertise.* AKM tools should consider that design reasoning is rather ad-hoc and not a rational process all the time.

Some of these challenges have helped us formulate the requirements for our bottom-up AKM approach (cf. Chapter 3). We have considered some of the directions suggested by Capilla et al. as guidelines for building the tool (ADeX) presented in this dissertation.

## 2.2.2 Architectural design decisions

The very first mention of design decisions was in 1980 (from the list of publications in our review list). During that time, software architecture, as a field of study, had not yet been well established, and the reference to design decisions was in the programming/source code context. In [P30], the author represented a design decision using an applicability predicate (antecedent) and a predicate expressing the desired qualitative effect (consequent). An example decision in a program is represented as:

**Decision:** *replace recursion by iteration*

**Antecedent:** the given program description is a linear recursive function

**Consequent:** the new version is equivalent to the given one and is an iteration using a fixed number of new auxiliary variables.

The author also showed how to represent the hierarchical specification of design decisions using composition rules based on formal logical representations.

Next, in 1990, the authors of [P31] argued that design decisions (encapsulation vs. interleaving, generalization vs. specialization, or data variables vs. procedures) could be detected in the existing code using reverse engineering. They also highlighted that once such design decisions are identified in code, they must be documented to support software maintenance and reuse.

Furthermore, during the 90s, when building software through the transformation of models was popular, the authors of [P32] proposed to map these models through design decisions and to ensure traceability across models. They presented the first tool (traceability support system as shown in Figure 2.5) that allowed programmers to capture the problem, alternatives, design decisions, the rationale behind those decisions, and the links between the source and the target models.

During the same period (early 90s) when Issue-Based Information Systems (IBIS) were becoming popular, many Software Engineering (SE) approaches were proposed based on IBIS. The authors of [P33] also used the IBIS representation as a method for capturing design argumentation. They introduced a model to help programmers answer questions such as “*Why certain decisions were made?*”, “*What alternatives have been explored?*”, and “*Will this change violate any design constraints?*”.

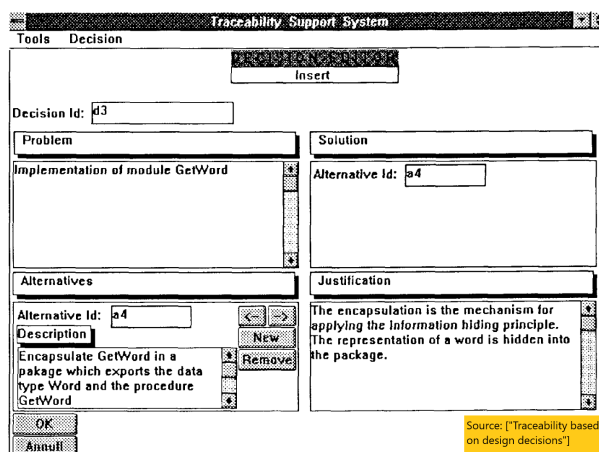


Figure 2.5: A traceability support system



Requirement	Description
Architectural review	which AK elements have been added or modified?
Review concerns	for a quality concern, which are the associated knowledge elements?
Evaluate impact	if there is a change in an element, what are the impacted elements (decisions, quality attributes, AEs)?
Get a rationale	for an AE, trace back to its related decisions.
Study the chronology	understand the sequence of design decisions over time.
Add a decision	manually or automatically integrate decisions to other AEs.
Cleanup the system	removing a decision should be reflected on associated elements.
Spot the critical stakeholder	stakeholders who have the most “weight” on a decision and those who will be affected if that decision changes.
Clone AK	for AK reuse
Integration	with multiple systems

Table 2.1: Requirements captured by Kruchten et al. [P1] for managing an AK repository

Kruchten in this seminal work [P34] stated that “the documentation produced during the architectural design is captured in two documents: a software architecture document, organized by the 4+1 views, and a software design guideline, which captures (among other things) important design decisions that must be respected to maintain the architectural integrity of the system”. To document these design decisions, the authors of [P35] proposed to use a tree structure. Wherein, each node contains “concerns and constraints, the solution to address the problem (decision), new concerns raised by applying a decision, and references to other decisions”. Kruchten in [P36] also highlighted that a software architect must be able to communicate design decisions to the involved stakeholders effectively and should consider their decisions as well. He further went on to say that an architect must “*make some decisions very quickly, based on experience and ‘gut feelings’ rather than pure, thorough analysis*”. [P37] also suggested that the ‘soft-skills’ and external influences are important in the socialization of decisions.

To understand what an AK repository should contain and how to use it, Kruchten et al. [P1] presented a list of requirements, as shown in Table 2.1. They also argued that since “the burden to capture assumptions and decisions outweighs largely the immediate benefits”, we should aim to *automate the collection of decisions and their rationale*. Kruchten et al., in that publication also showed how to represent decisions using a graph and that the requirements in Table 2.1 correspond to certain operations on that graph.

Kazman correlated ADDs to quality attributes in [P38] (“the quality attributes of a system are dictated by its architectural design decisions”). In this work, he presented the Cost Benefit Analysis Method (CBAM) to model the costs and the benefits of ADDs. Figure 2.6 shows the context of the CBAM which links costs to ADDs. According to that context, a design decision addresses quality attributes and stakeholders can analyze the cost-to-benefit ratio of investing in a

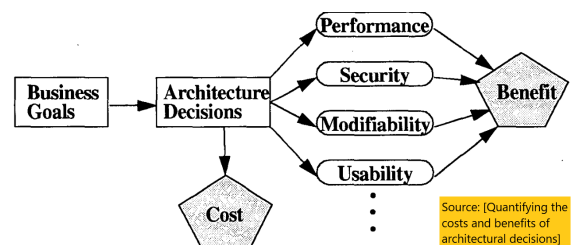


Figure 2.6: The context of CBAM

specific decision. CBAM was further revised to incorporate an iterative elicitation process with a decision analysis framework [P39]. Another publication [P40] also investigated the economic perspective of capturing ADDs and presented a “structured intuitive model for product line economics” for product line decision-making. Furthermore, to compute the savings in effort, [P41] proposed a cost-benefit framework for ADM. By analyzing the differences in an architectural refactoring activity, this framework correlates the developer effort to the change in coupling.

The very first explicit mention of the statement “*architecture of software is a collection of design decisions that are expensive to change*” was in [P42] (in 2001). The author of that publication, Ran argued that the “expensive to change” decisions are those decisions on which most other decisions depend on (indicating that design decisions are interrelated). Bosch in 2004 also emphasized that “*software architecture is, fundamentally, a composition of architectural design decisions*” and “*designing a software architecture can be viewed as a decision process*” [P43]. Researches in software architecture community attribute the paradigm shift in how we view software architectures to this work by Bosch as well. Furthermore, since knowledge about design decisions are lost over time and the changes to the software system can violate earlier decisions, he pointed out that design decisions should be represented as first-class entities in software architecture. Jansen and Bosch in [P44] also pointed out that a tool that supports the evolution of software architecture must: (a) support multiple views, since design decisions affect various architectural concepts and (b) support the addition, modification, and deletion of those design decisions. However, given that design decisions are not explicitly documented (which is also emphasized in this dissertation), Jansen and Bosch [P45] proposed “Architectural Design Decision Recovery Approach”, which is a bottom-up approach to recover design decisions. By comparing different versions/releases of a software system, the architectural deltas are extracted which reflect on the decisions made by architects.

Van der Ven and Bosch showed that *commit messages in source code repositories are yet another source wherein design decisions are implicitly captured by architects and developers* [P46]. They analyzed 100 commits (mostly related to ruby gem files) and concluded that medium-level decisions could be identified in commit messages. Medium-level decisions include for instance the selection of specific components or interaction between components. Another *potential data source for reverse engineering design decisions is the bug reports of software systems* [P47].

The proposal to extend the 4+1 view model with the decision view was made in [P48]. In that paper, Dueñas and Capilla, also capture the following requirements for managing ADDs:

- *The system should have multi-perspective and multi-user (groupware) support.*
- *ADDs should have visual representation so that they can be easily understood.*
- *ADDs in large systems should have some classifications (hierarchy, abstraction).*

On the abstraction level of design decisions, the author of [P49] distinguished between strategic (e.g., architectural style, CBSE standard, or application framework) and tactical decisions (e.g., design pattern, refactoring, or programming idiom). He argued that strategic decisions have a far-reaching impact on the implementation and tactical decisions have a localized effect on the software system. To capture such design decisions a template called “architectural decision description template” was proposed in [P50]. This template includes concepts such as Issue,

Decision, Status, Assumptions, Constraints, Arguments, Implications, and Related decisions. Another method to capture design decisions is using mind-maps. Using a mind-mapping tool, architects can capture the internal structure of ADDs (including their dependencies) in the form of diagrams [P51]. A template-based approach that was evaluated with students in the context of service-oriented design is presented in [P52]. It contains the basic concepts related to ADDs but does not capture relationships between ADDs and software artifacts. On the other hand, [P53] showed how to link structural and technological decisions documented in decision templates with requirements and the corresponding architectural models.

In 2007, Clerc et al. [P54] based on a survey found out that even though the idea of representing software architecture as a set of ADDs was not completely adopted in practice, the concept of ADDs had gained importance among practitioners. These ADDs address system behavior, the interaction between components, system deployment, evolution, and non-functional properties of a software system [P55]. To reduce the amount of work required for practitioners to capture these ADDs, [P56] presented a straightforward approach comprising of three steps: flag, filter, and form. As the name suggests, when architects encounter a design decision, they can flag it; then using filtering they can find relevant decisions in a project, and lastly, forming means to create a decision entry (decision, priority, category, status) based on the decision model.

Selecting an architectural style or a pattern is an essential type of ADD. [P57] emphasized that point, and linked patterns to high-level ADDs. [P58] suggested that ADD recovery can be improved by focusing on those decisions related to the application of specific architectural patterns. In [P59], a pattern model is linked to the component meta-model, and the selection of a pattern and the corresponding components implicitly captures that decision.

Heijstek et al. [P60] conducted interviews with 47 participants and found that neither diagrams nor textual descriptions were significantly more efficient for conveying ADDs. However, it should be noted that the diagrams used in their study was either component or class diagrams. We believe that it would have been more interesting, if those authors had conducted similar studies with, for instance, decision graphs instead of classical design diagrams.

To document design decisions made at the source code level, [P61] introduced Java annotations to mark operations of reused components. These annotations within components can then be used to check usage compliance when they are reused.

[P62] presented the results of a systematic mapping study (from 2005 to 2011) to confirm with our findings that there has been a growing interest in the topic of ADDs. They highlight that there exists little research in the area of GDM and uncertainty in decision making. We found eight more publications specifically discussing GDM since their study was published in 2014. And regarding the topic of uncertainty, in our research department, we have recently started investigating this topic (cf. [69] and [70]).

Miesbauer and Weinreich [P63] conducted interviews with nine experts from six different companies and listed down very interesting findings (which are also highlighted in other publications discussed in this chapter) that are relevant for this dissertation:

- *Non-existence or ban decisions are not captured.*
- None of the participants focused on process and tool decisions.

## 2. Foundations and related work

- Decisions can be classified into organization, project, architecture, implementation, and deployment decisions.
- Places for documenting design decisions include: *source code, meeting minutes, project diaries, issue tracking systems, and wikis.*
- Even if critical design decisions are documented, their rationale are not captured.
- *Personal experience and preferences have a significant influence on the ADM process.*
- Previously made decisions have a high influence on new decisions (they are inter-related).
- Requirements drive design decisions.
- While documenting design decisions may be costly, wrong decisions are expensive, too.

### 2.2.2.1 Formal representation of architectural design decisions

In this subsection, we present those publications that focus on the models of ADDs. Some of these publications refer to the model as a meta-model, however, we refer to them as the domain model for capturing ADDs (cf. the Object Modeling Group’s modeling layers).

Jansen and Bosch [P64] present an approach called Archium and also its model which represents design decisions as first-class entities. They emphasize that architecture is the composition of a set of ADDs. Furthermore, they also explicitly define design decisions, and we use the same definition in this dissertation:

“A description of the set of architectural additions, subtractions and modifications to the architecture, the rationale, and the design rules, design constraints and additional requirements that realize one or more requirements on a given architecture.”

The conceptual model for ADDs as presented by Jansen and Bosch is shown in Figure 2.7. An issue can be resolved by multiple solutions and the decision is to choose an alternative by making trade-offs among the alternatives. Making such a decision results in the architectural modification which influences other existing/future requirements and decisions. Based on this model, the authors present the Archium approach and with examples, they show how to capture design decisions in a given context.

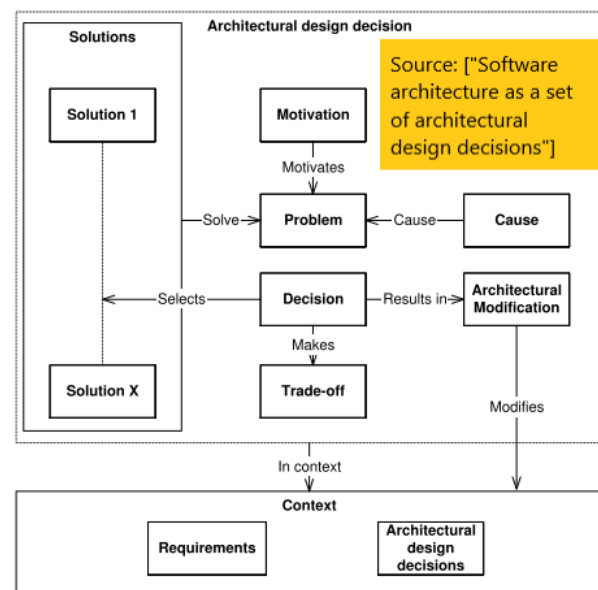


Figure 2.7: The conceptual model of ADD used in the Archium approach

In [P65], the authors propose an ontology that is composed of architectural assets, ADDs, stakeholder concerns, and an architectural implementation roadmap. Since this was a position paper, not much details about the ontology (its creation and maintenance) are presented. However, Kruchten et al. [P2] present a detailed description of an ontology for AK representation with the focus on design decisions. Kruchten et al. argue that “AK consists of architectural design as well as the design decisions, assumptions, context, and other factors that together determine why a particular solution is the way it is.” In short, “ $AK = Design\ Decisions + Design$ ”. Kruchten et al. also emphasize the fact that AK (design decisions and their rationale) is tacit, remains in the heads of architects, and are usually lost since they are not explicitly captured. And hence, it becomes difficult to trace the reasons for already made design decisions. They also hint that there is a need to “*automate the collection of rationale (or of decisions, or both)*”. In the ontology presented by Kruchten et al., they provide a taxonomy of design decisions and capture their relationships. The definitions of different types of design decisions are presented in Table 2.2. These definitions are important for this dissertation, since, they are referred to in Section 5.3 while presenting the process of automatic extraction and classification of design decisions. They also discuss the following types of relationships between design decisions: constraints, forbids, enables, subsumes, conflicts with, overrides, comprises (decomposes into), is an alternative to, is bound to, is related to, and depends on.

The authors of [P66], also argue that despite the importance of ADDs (capitalizing on the previous ADDs to provide the foundations for learning and training), ADDs remain tacit and are lost over time. In the model they propose, they highlight the changes made to the design of a system. They provide a vocabulary for those operations that change the design: addComponent, addConnector, deleteQualityRequirement, addFunctionalRequirement, deletePort, and so on. Such operations applied to the design of the system can be versioned to preserve the history of

Design decision	Description
1. Existence decisions	one element/artifact will positively show up, that is, will exist in the system’s design or implementation.
1.1. Structural decisions	lead to the creation of subsystems, layers, partitions, components in some view of the architecture.
1.2. Behavioral decisions	are related to how the elements interact together to provide functionality or to satisfy some quality requirements, or connectors.
1.3. Bans or non-existence decisions	state that some element will not appear in the design or implementation. They are not traceable to any existing artifact.
2. Property decisions	state an enduring, overarching trait or quality of the system. They can be design rules or guidelines or design constraints, as some trait that the system will or will not exhibit.
3. Executive decisions	decisions that do not relate directly to the design elements or their qualities, but are driven more by the business environment, and affect the development process, the people, the organization, and to a large extent the choices of technologies and tools.

Table 2.2: A taxonomy of design decisions: presented by Kruchten et al. in [P2]

ADDs. In our work, we detect such operations by applying Natural Language Processing (NLP) on the textual description of issues in an IMS.

Zimmermann et al. [P67] present a model which includes concepts such as ADLevel, ADTopic, ADD, ADAAlternative, and ADOOutcome. Based on this model, they propose a framework for collaborative ADM; especially in the context of enterprise architectures. [P68] also presents a model for AKM and the authors argue that the critical elements of an AK model are ADDs and their rationale. They highlight that internalizing AK helps architects to avoid sub-optimal solutions and to learn from their mistakes. An approach called ADVERT [P69] helps architects to document ADDs, their rationale, and links ADDs to the triggering requirements and the implementing Architectural Elements (AEs). The authors of [P70] extend the model presented in [P67] by including links between ADOOutcome and software artifacts represented by the ADArtifact entity. To indicate the impact of an alternative on the architectural artifact, they also introduce wouldImpact-traceability link between ADAAlternatives and ADArtifacts.

An AK model called CORE [P71] captures the links between ADDs, AEs, stakeholders, and processes. This model also characterizes ADDs using concerns, decision topics, and alternatives. De Boer et al. in this publication emphasize that ADDs are interrelated and their relations typically form a tree structure (from abstract to concrete decisions). Another model presented in [P72], captures ASRs, ADDs, alternatives, and rationale along with their relationships. [P73] also captures an extensive AK model comprising of concepts ranging from stakeholders, their concerns, requirements, decisions, to architectural artifacts. This model explicitly emphasizes the involvement of multiple stakeholders during ADM. The authors of [P74], in their model, explicitly link ADDs to requirements. They also discuss two types of ADDs (recurring and project specific) and their effects on requirements and their prioritization.

To link and to capture the traceability from ADDs to AEs in Unified Modeling Language (UML) diagrams, [P75] proposes a UML profile for capturing ADDs and an associated profile for representing Non-Functional Requirements (NFRs). Such profiles ensure the treatment of ADDs as first-class elements in the design of a system. [P76] also proposes architecture-specific decision types (virtualization, replication, single instantiation, etc.) and link them with UML diagrams. UML's object constraint language is then used to capture constraints of those decision types. The *link between ADDs and AEs* is explicitly shown in the model presented in [P77] and [P78]. The links include cardinalities of 1:1, 1:n, and n:1 relations between ADDs and AEs. Using these links and a decision constraint graph, impact analysis can be performed to identify those AEs affected by changed ADDs. Another model that explicitly links ADDs to architectural solution-elements is discussed in [P79]. For documenting ADDs, authors of [P80] present a model and a graphical modeling notation called "Maps of Architectural Decisions". This model includes concepts such as concern, connector, solution, requirement, decision-maker, pros, and cons. These concepts can also be tagged with elements such as defined, chosen, solved, (in)feasible, etc.

[P81] emphasizes on representing the relationships between ADDs as well as the NFRs addressed by those ADDs and AEs implementing those ADDs as a network of design knowledge. The model presented in [P81] includes relationships that were proposed by Kruchten et al. in [P2]. In a related publication [P82], a Concept-Requirement-Decision tree is used to organize architectural topics hierarchically. Similarly, a design map is used in [P83] for capturing ADDs (selection of architectural styles) and tracing those ADDs to specific NFRs. With the focus on facilitating

traceability from requirements to design, [P84] links concepts from the goal-oriented modeling language (goal, soft goal, task, and resource) to the concepts in an ADD model (decisions, alternatives, and rationale).

The authors of [P85,P86] also capture the relationships between NFRs, ADDs, and AEs in their tactic traceability information models. They argue that “*extracting the traceability links by annotating information from existing architectural documents can reduce the time of information retrieval from documents*”. The idea is to recover decisions and their traceability links from documents by training a ML classifier (*a bottom-up approach*). Even though a very important work in the context of this dissertation, unfortunately, the publication [P86] is only a proposal and the technical details for realizing such an approach are missing.

In the context of software Product Line Engineering (PLE), Capilla and Babar [P87] show how the models of the PAKME and ADDSS tools can be merged to support the concepts of PLE. This integrated model captures the dependencies between ADDs and the variability rules in the associated feature models. Furthermore, the need to capture another related concept – *vulnerability* of ADDs along with the rationale is discussed in [P88].

In the model presented by Tang and Van Vliet [P89,P90], they emphasize the concept of design constraints. They also categorize those constraints into requirement-, quality requirement-, context-, and solution-related constraints. These constraints influence and drive ADDs made by architects during the design phase. To support the evolution and maintenance of AK, Che and Perry present a so-called Triple View Model (TVM) [P91,P92]. This model captures the What (element view), Why (intent view), and how (constraint view) of ADDs.

Zimmermann has been a notable author who, over the years, has contributed to ADM in the context of service-oriented architectures (SOA). In [P93], he presents an ADM framework for SOA systems. The decision templates used within that framework can assist the selection of a Software Architecture (SA) that supports runtime models. In another paper [P94], Zimmermann et al. investigate the use of ADD models for the microservice architecture. In this work, they present (a) a taxonomy of the areas of microservice design, (b) the involved stakeholders, (c) the use cases for which microservice decision model is helpful, and (d) the model elements that are part of the decision model (system components, technology options, etc.). Zimmermann with his colleagues, in [P95], emphasize the need to capture the context of a decision explicitly. Understanding the context of ADDs is critical for enabling recommendations and reuse in a similar context. Hence, they propose a context model comprising of five concepts (organization, product, stakeholder, development method and technology, and market and business).

In a series of publications [P96–P98] in 2015, Zdun and his colleagues present the tool called CoCoADvISE and the so-called reusable decision model. CoCoADvISE uses a knowledge base to support ADM in the context of software ecosystems. The knowledge base maintains design patterns, tactics, quality attributes, decision drivers, decisions, their relationships, and domain experts’ feedback on the documented ADDs. CoCoADvISE uses the knowledge base to generate specific recommendations and to produce a questionnaire for making new ADDs.

The technology related decisions are one of the most often made decisions. Soliman et al. [P99] extend the AK model to capture technology decisions (with concepts such as architecturally significant technology aspects, their features, architectural aspects, benefits, and drawbacks).

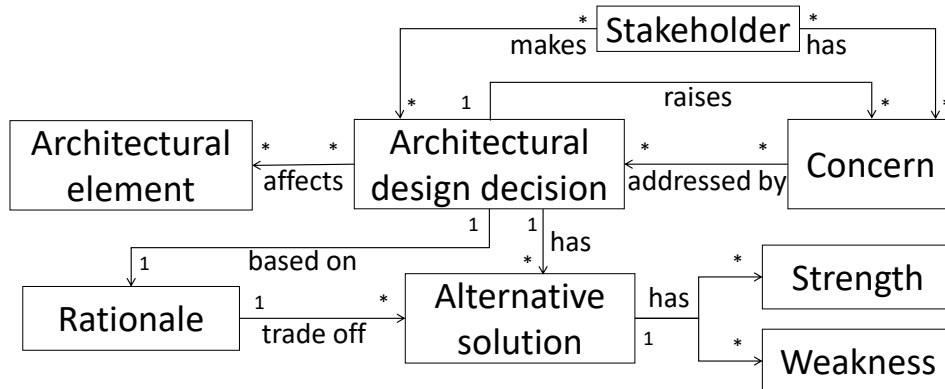


Figure 2.8: A conceptual model capturing ADD and its associated concepts

The authors of [P94] and [P100] also extended the AK model to reflect the sustainability of ADDs. The notion of sustainability of ADDs has been elaborated by Zdun et al. in [P101]. They state that the ADD sustainability comprises of (a) “the time period when the right and relevant decisions remain unchanged”, and (b) “the cost efficiency of required changes to those decisions”. To reduce AK documentation effort, Zdun et al. propose to capture only significant ADDs and their relationships using configurable AK templates (lean approach for documentation).

To help stakeholders measure the quality of knowledge captured across multiple decision models and for reuse thereof, [P102] presents a set of metrics to capture stakeholders’ goals.

In [P103], Shahin et al. compare nine AKM tools and their models and state the following:

- ADD models treat architectural design as an ADM process
- ADD models have consensus on capturing rationale, constraints, and decisions’ alternatives
- Tools do not exist for all the ADD models, some of them use text templates for capturing and documenting ADDs
- *ADD personalization is a desired feature that is currently missing in most of ADD tools*

The main takeaway for the readers from the analysis of the aforementioned models is that the core of any AK or ADD model is the concept of ADD and its rationale. The quality concerns of stakeholders are the main drivers for ADDs [P104]. This is also evident in the conceptual model of architecture description prescribed by the ISO/IEC/IEEE 42010 standard [4]. The model shown in Figure 2.8 summarizes the main concepts (based on the already discussed models) that are relevant to this dissertation. In essence, Stakeholders have concerns and architects and developers address those concerns by making ADDs. Making an ADD affects one or more architectural elements as well as may raise new concerns. An ADD should be made by considering multiple alternative solutions and by analyzing the trade-offs between their strengths and weaknesses, which is the rationale for making that ADD.



### 2.2.2.2 Architectural design decisions - tools

The first publication (in our literature review) that explicitly focuses on capturing and ordering ADDs was in the year 2000 [P105]. This paper presents Argo/UML that targets the cognitive tasks (decision ordering in a to-do list, ADM, task-specific design understanding) during software design. Supporting designers to make “good” ADDs is important since those ADDs strongly influences the implementation and maintenance effort.

In a series of publication from 2006 to 2008, Capilla et al. [P106–P112] present a web-based tool called Architecture Design Decision Support System (ADDSS) to record and manage ADDs. The model behind ADDSS which represents a design decision as a first-class entity is shown in Figure 2.9. Design decisions are mapped to the requirements of the system through stakeholders and affect the architecture of a software system.

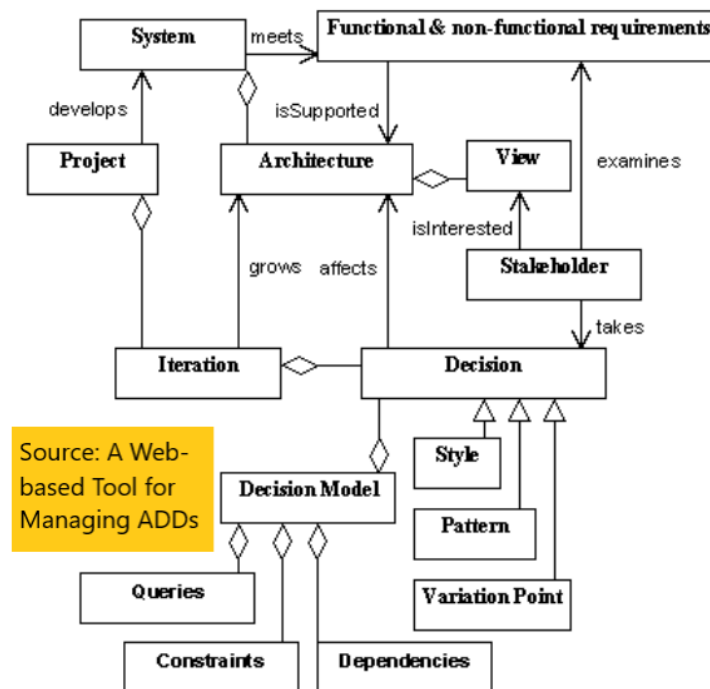


Figure 2.9: The conceptual model of the architecture design decision support system

Capilla et al. discuss that ADDSS<sup>1</sup> supports the following features:

- Groupware support (web-based tool)
- Multiple projects and architectures
- Different categories of users and permissions
- Architecture visualization (architecture models can only be uploaded as images)
- Multi-perspective support (different views on web pages)

<sup>1</sup>The link provided to access ADDSS is no longer available

## 2. Foundations and related work

---

- A visual representation (a simple UI to list design decisions)
- Gradual formalization (a chronological visualization of the decisions)
- Architecture iterations (versions, related to gradual formalization)
- Patterns and styles (with add/remove options)
- Design decisions support (rather naive: linking design decisions to styles and patterns)
- Dependencies between decisions (detailed categorization of dependencies between decisions is part of their future work)
- Alternative decisions (add/remove to a list)
- Support for (non-) functional requirements (links to decisions)
- Architecture documentation (generate reports)
- Basic customization mechanisms
- Timers to measure the capturing effort of ADDs
- RSS feeds to notify updates
- Complexity control (not clear how)

Based on his experience with ADDSS and comparing other AKM tools, Capilla in [P113] suggests the following improvement areas for the next generation tools:

- New tools should integrate all aspects of the design process (e.g., with existing architectural specification and modeling tools).
- New tools must support creating concrete links between decisions and resulting models (as well as embedded facilities for design rationale).
- New tools that help architects in capturing ADDs should *reduce their intrusiveness*.

With the focus on visualization of ADDs, Lee and Kruchten [P114,P115] present a tool for ADDs exploration and analysis. Using this tool, users can create, modify, remove, and view ADDs and their relationships. ADDs are visualized using a directed graph (nodes represent ADDs, and the directed edges represent the impact-relationships between them). Another visualization within this tool represents the historical view of a set of ADDs which captures their evolution as well as their status using different symbols.

Lee and Kruchten [P116] present another tool that filters AK from tags/annotations in the source code. A software-package in their tool also extracts AK from emails. In this work they differentiate between three approaches for capturing ADDs:

1. Formal elicitation: gathering software decisions in an explicit and structured manner (which uses formal modeling)
2. lightweight top-down approach: focuses on the early design phases, to capture minimal/necessary information about ADDs.

3. **lightweight bottom-up approach**: capturing ADDs that are documented within different artifacts generated during Software Development Lifecycle (SDLC).

It should be noted that, according to these three approaches mentioned above, the system (ADeX) presented in this dissertation follows a lightweight bottom-up approach. In our approach, we automatically extract AK from tools such as MS Project, Enterprise Architect, Jira, and Github as well as process them further using ML and NLP techniques.

Another visualization-centric tool is presented by De Boer et al. [P117] and refer to the tool as ontology-driven visualization (ODV). ODV uses table and matrix representations of decisions and quality attributes. ODV is used to support reviewers to assess the product quality (with trade-off analysis and if-then analysis scenarios).

An extension to a tool named Compendium to support the visualization and exploration of ADDs and their underlying rationale is presented in [P118]. Compendium is a semantic hyper-text concept mapping tool and supports argumentation-based approach for ADM. Within this tool, each compendium node can be represented as a decision, note, question, answer, or pro/con argument. Using those nodes, different visualizations (e.g., dependencies between ADDs) can be created to support collaborative ADM [P119]. Furthermore, Shahin et al. evaluate these visualizations in a controlled experiment with 10 participants [P120]. They show that visualization diagrams of ADDs improves architects' understanding of the existing architecture. In another experiment with 21 subjects, Shahin et al. also indicate that "experienced participants benefited more from ADDs in comparison with less experienced ones" [P121].

In [P122], the authors present a tool to integrate a decision management system (Architectural Decision Knowledge Wiki) and a UML-based modeling tool (IBM Rational Software Modeler) to closely *link decisions to the structural and behavioral elements in the models*.

Chen et al. [P123, P124] focus on customization features in their Architecture Design Decision Management (ADDM) tool. Within this tool, they provide users of the system the *flexibility to adapt the ADD model to organize their AK in a better manner*.

A semi-automatic approach to capture traceability relations from design decisions to architectural elements and the code base can be achieved using the tool called LISA [P125]. When developers are implementing a selected active design decision, modification events are automatically created. These events include information about the impacted implementation and architectural elements. These modification events are maintained in LISA along with the link to the corresponding active design decision which can be reviewed thereafter.

A bottom-up approach that uses domain ontologies is presented using a tool called TREx in [P126]. This tool uses domain ontologies to annotate text with architectural topics. Based on those annotations and predefined rules, TREx tags statements as either design decisions or design structures. However, creating these rules seems to be an effort-intensive manual task that needs to be performed by domain experts.

A system called Design Practice Streams (DPS) that uses video recordings of design meetings to help architects reflect on the made ADDs is presented in [P127]. DPS records the sketches drawn on a whiteboard as well as produces a textual transcript of the voice recordings. After the design meetings, architects can search for topics in the transcripts or select a region on the

whiteboard and navigate to specific segments within the video. [P128] also proposes the idea of using a video wall for collaborative ADM. In that idea paper, they indicate that one can analyze the meeting discussions (multi-touch screen for the screenplay, audio, and video) with text and argumentation mining.

Another interesting tool that uses design meetings recordings is called Design Verbal Interventions Analysis (DVIA) [P129–P131]. In this tool, the meeting recordings are used in combination with verbal intervention analysis model to classify and map segments within transcribed meeting logs to an ADD model (which comprises of issues, orientations, clarification, explanations, disagreements, constraints, assessments, choices, and assumption elements). However, it is not clear which parts within their approach is manual/automated<sup>2</sup>. The process of identifying the intent behind a segment within the meeting transcript and mapping those intents to the decision model seems to be manual and has not been elaborated within the paper. They do mention that ADD recovery is difficult, costly, and inefficient and requires effort for transcription, identification, and categorization of decision topics.

For persisting and choosing adequate design patterns from a knowledge repository and supporting architects during the development phase, an AKM tool called Architectural Development using Architectural Knowledge (ADUAK) is presented in [P132]. Another repository-based tool called ArchiTech allows domain experts to create ADDs which are prominent in a specific domain and link them to NFRs [P133]. By capturing the above information, one can reuse them across projects in the same domain by suggesting ADDs to address specific NFRs. The authors of [P134] also propose maintaining a repository of reference architectures and styles. Then, once the system requirements are captured in their tool, stakeholders can prioritize the requirements, and finally based on the priorities, stakeholders can get recommendations about choices. A similar approach called ADMD3 is presented in [P135], where the idea is to annotate recurring ADDs using specific checklists (decision-specific questions) and to persist them for future reuse.

An extensive tool to support architects to reason about ADDs is presented in [P136]. The tool referred to as Architectural Design Decision Support Framework (ADvISE) is an Eclipse plug-in which supports creating reusable ADD models. Using the Question, Option, Criteria (QOC) method, this tool allows creating questionnaires for making ADDs. The View-based Modeling Framework (VbMF) integrated with AdvISE provides the component-and-connector view-model and links the model elements with decisions in ADvISE. The tool also supports considering uncertainty using basic fuzzy logic based inference mechanisms.

A tool based on the repertory grid technique called RGT tool<sup>3</sup> is presented in [P137, P138]. This tool allows users to capture ADDs, prioritize concerns, and analyze ADDs in a group setting using different viewpoints. The viewpoints include (a) chronological viewpoint (changes to ADDs over time), (b) dependency viewpoint (relationships between ADDs), and (c) group viewpoint (stakeholders' concerns, alternatives, ratings for alternative-concern pairs).

Manteuffel et al. have developed a plugin called Decision Architect<sup>4</sup> (DA) for the Enterprise Architecture (EA) system [P139]. The model of DA is based on the ISO/IEC/IEEE 42010 model

---

<sup>2</sup>There is no mention of the executability of the DVIA tool for testing.

<sup>3</sup>The link provided for the tool in [P137] is not available.

<sup>4</sup><https://archive.codeplex.com/?p=decisions>

along with the addition of Relationship and Relationship Type concepts. DA allows users to create, edit, delete, and view ADDs, it's associated concepts and relationships between them. DA provides five different viewpoints to support different concerns (e.g., decision relationship view shows relationships between ADDs). A very powerful feature of such a plugin is that the EA system provides the necessary models (e.g., from requirements and design) which can be linked with the plugin's model. Given that the EA system is widely used in industry, such a plugin addresses the challenge of applying and (re)using AK during architects' day-to-day activities.

Another Eclipse plugin called DecDoc allows users to capture ADDs and link them to artifacts such as requirements (from an AKM tool - UNICASE), architectural elements, and code [P140]. This tool focuses on supporting collaborative and incremental documentation of ADDs.

Even though there is an abundance of ADD models and tools for capturing of ADDs, they are not adopted in practice and ADDs are seldom documented [P139]. Like many other publications discussed before, [P141] suggest the following reasons for the lack of adoption:

- The absence of *industrial applicability requirements*
- Only marginal support of brownfield development (*bottom-up approaches*)
- Insufficient consideration of the *overhead* during the documentation of ADDs
- Simplified or often missing *perspective on the evolution of systems and ADDs*
- Lack of tool-support and *integration with commercial tools*

### 2.2.3 Architectural design rationale

Even though the term design rationale was not included in the search query, we found ten publications specifically focused on ADR. Among them, five publications are from 2004 to 2008. During this time, it should be noted that rationale-based/driven approaches were also popular in the software engineering research community.

To systematically manage design rationale, Burge and Brown [P142] present a tool named InfoRat. The model behind this tool is quite straightforward as shown in Figure 2.10. In this model, the concept requirement can be broken down into goals and sub-goals. Goals can then be satisfied by one or more alternatives. The rationale behind an alternative is represented as claims, for or against each alternative. Once information is captured using

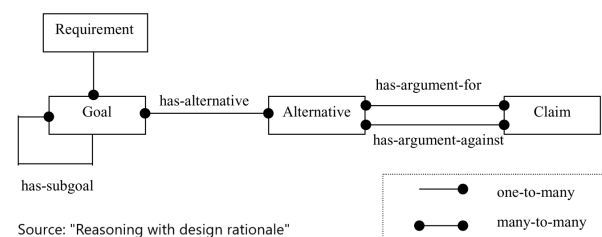


Figure 2.10: An architectural design rationale model by Burge and Brown

InfoRat, inferences can be made to find inconsistencies and completeness of goals. Continuing her work on design rationale management, Burge with her coauthors presented SEURAT (Software Engineering Using RATIONale) system with the aim to *integrate ADR capture and visualization into standard SE practices* [P143,P144]. Since architects do not document their decisions as a separate process, SEURAT was integrated into Eclipse IDE, so that it is tightly integrated with

the design and development activities and the capture and the use of rationale becomes part of a standard process. By using SEURAT for capturing rationale for design choices and applying semantic inferencing, authors argue that the impact of decisions on a software system can be determined.

Another extension to the SEURAT tool called SEURAT-Architecture is presented in [P145]. This extension uses a predefined library of patterns and NFRs to guide architects during the selection of architectural patterns. Each recommended pattern for a concern can be considered as an alternative and the reason as to why a pattern was recommended acts as the rationale.

In [P146], the authors present a web-based tool for managing ADR. They indicate that an ADR should comprise of: (a) a description of issues addressed prior to the decision; (b) a list of the considered alternative solutions; (c) the criteria used in the selection of an alternative; (d) the arguments used to support/oppose each alternative; (e) the final decision;

An approach for documenting ADR is presented by Falessi et al. in [P147]. The so-called value-based design decision rationale documentation (VB DDRD) approach focuses on documenting only those set of required information depending on the documentation purpose (e.g., what-if analysis and avoiding design erosion). The ADR information includes concepts such as issue, design decision, status, assumption, constraint, etc. Falessi et al. suggest capturing minimal/required information (lean models) to describe ADDs and to reduce the burden and effort during the design process in an agile development environment [P148].

Another tool that focuses on integrating ADR as part of the design documentation process uses a template-based approach [P149]. This work shows how to support architects in retrieving ADR using a tool. However, many of the tool's screenshots seem to be manually edited.

An online “intelligent software architecture rationale capture system” presented in [P150] allows architects to manage ADR collaboratively. The tool helps architects understand the relationships between ADR, requirements, and architectural elements during SDLC.

Tang et al. [P151] present practitioners' view on the value of ADR; how they document and use design decisions. Their study shows that practitioners are aware of the importance of ADR and they use them more often than they document them. Apart from discussing the need for improved context-specific ADR tool support, they also reflect on biases in decision making. They highlight that architects focus on “good news” rather than on “bad news”. That is, architects tend to discuss the benefits of new technologies without focusing on their negative aspects. Such *biases have to be studied since their awareness can help architects to be objective during ADM.*

### 2.2.4 Architectural decision making

The act of making a design decision by analyzing the trade-offs among alternatives to meet the desired criteria and then selecting an alternative is referred to as the ADM process (cf. [P152] for a similar definition). Over the years, there have been many discussions about how architects go about such an ADM process. That is, do they follow a systematic approach by listing and prioritizing alternatives before making a decision (known as Rationalistic Decision Making (RDM) approach)? Or, do they make a decision based on their experience and gut feelings

(known as Naturalistic Decision Making (NDM) approach)? Furthermore, there are also few studies which explore those factors that influence an architect's decision-making process, such as, time, budget, cognitive biases, and architectural assumptions. In this section, we discuss those publications in our review list that reflect on the ADM processes as well as the factors influencing those processes.

The very first publication in our review list that highlights that architects follow the NDM approach was published in 1996. Even though [P153] has not been cited by anyone and has not received any attention, the author states an important point: "the set of design decisions is handled *intuitively*, and also the set of alternatives for each of the design decisions is strongly influenced by *experiences* from earlier designs".

For ADM, authors of [P154] present a method developed at Siemens called "system architecture analysis". This method includes a set of evaluations to ensure that (a) different alternatives were considered, (b) their pros and cons were discussed, and (c) design decisions and their inter-dependencies were analyzed.

Zannier and Maurer [P155] present the structure of their qualitative study. They elaborately discuss the definitions of RDM and NDM, as well as introduce factors such as expertise and mental modeling affecting ADM. In their subsequent work, Zannier and Maurer [P156] conducted twelve case studies by interviewing software developers and mentors in agile projects. They indicate that NDM dominates RDM and in NDM experience and intuition play a crucial role. They also found out that agile projects supported better communication and debate about alternatives between architects and developers as compared to traditional software development projects [P157]. They also suggest that, if a design problem is well-structured, architects tend to use RDM, whereas if it is ill-structured, then NDM is preferred [P158]. In another publication [P159], Zannier and Maurer based on their interviews conclude that *architects do not always strive for optimal alternative and do not always consider alternatives* (a key to RDM). However, they indicate that "alternatives are considered more often in groups (in casual conversations)" and ADM is a highly cognitive and a social process.

In [P160], the authors summarize the commonly used ADM strategies which could be used for selecting a component given the alternatives that meet specific stakeholders' constraints. They provide an overview of the strength and weaknesses of Analytic Hierarchy Process (AHP), simple multi-attribute rating technique, utility theory, and weighted score method. However [P161] argues that companies (six Norwegian companies, eight interviews) do not use well-known ADM processes but rather use their own customized approaches. Based on an industrial case study, the authors of [P162] also confirm that architects do not follow any systematic ADM approach but instead follow informal and lightweight approaches. They mention that factors like time, money, and organizational practices influence ADM in organizations.

Tang et al. [P163] conducted a controlled experiment with two groups to understand the effects of design reasoning while addressing design concerns. They found that explicitly capturing the design rationale (for design decisions) improves the overall quality of the architecture and helps architects to backtrack the decisions. Tang et al. emphasize that explicit reasoning helps architects to avoid architectural assumptions during the ADM process [P164]. They conducted another study to understand how architects consider alternatives during ADM [P165]. In that

study they found out that architects do not explore many options and finalize their decisions as soon as they have good enough supporting reasons; indicating that architects exhibit satisficing behaviors (a characteristic of NDM). The idea of managing architectural assumptions is further discussed by Yang et al. in [P166]. Based on their interviews they found that even though the term architectural assumptions is not commonly used, architects frequently make architectural assumptions during ADM.

On the notion of applying past experiences for making new ADDs, [P167] reports that architects use their expertise to arrive at an ADD (especially in the early stages of design) without performing extensive [mental] search for an optimal alternative. Using experience or analogy to recognize similar patterns of design concerns is a characteristic of recognition primed decision-making which is an NDM approach.

A decision-making framework comprising of two levels (macro and micro) is presented in [P168]. The macro-level is used to analyze the design strategy using problem and solution orientations. Whereas the micro-level uses the decision mode and the decision strategy for analysis. The authors of [P168] argue that ADM is a creative exploration process which includes problem recognition and hypotheses testing. Another framework called GRADE (Goals, Roles, Assets, Decision, Environment) is presented in [P169,P170] to support architects while selecting a sourcing option (in-house, COTS, open source, and outsourcing). Another project called ORION also supports the selection of sourcing options by capturing decision cases in a knowledge repository [P171].

The authors of [P172] combine risk-based reasoning with the quality attribute model to support ADM. First, risks drive architects to select an architectural style to meet the quality goals, and the quality goals help to choose architectural tactics. Another framework that uses architectural tactics to select software components is presented in [P173]. The framework called COMPo-nents using ArChitectural Tactics (COMPACT) supports a collaborative component search (in a catalog using keyword search) and recommendation of tactics to suit NFRs.

In the context of agile development, [P174] presents a so-called responsibility-driven architecture to understand when, how, and who should make ADDs. Another decision-centric architecture design approach is given in [P175]. This approach follows a sequence of steps during a design process. First, an issue (architecturally significant requirement) leads to multiple candidate architectural solutions. Selecting a candidate solution reflects a decision and the pros and cons for the solutions become the rationale for that decision.

Lytra et al. conducted a multi-method study to identify 400 potential service-based integration patterns. Based on this study, they then proposed a frequent-items set method to investigate (based on the frequency of co-occurrence of decision points) which decision points may or may not coexist in the design decision space [P176].

Tofan et al. [P177] conducted interviews with 43 architects and suggested the following research directions. There is a need for (a) *addressing uncertainty in ADM*, (b) *better approaches for GDM*, and (c) *improved understanding of dependencies between ADDs and effort estimation to analyze ADDs*. The topic of uncertainty in ADM is being investigated by a colleague at our research department<sup>5</sup>. Even though the other two challenges are relevant, unfortunately, they are out of scope of our work, and we too consider them as part of future research.

---

<sup>5</sup>See: <https://www.matthes.in.tum.de/pages/fnxvsn1w9ck1>



In [P178], the authors model the ADM activities using the business process modeling notation. The activities include motivating a decision, specifying candidate alternatives, selecting the best alternative, and finally collaborative review and approval. Another process-based approach is presented in [P179] which proposes an ADM process using a tag-based traceability system. It supports collaboration among architects, notifies them about changes, and includes feedback loops to improve the ADM process. However, being a short paper, the authors do not detail the approach, for instance, how the tag-based approach works for maintaining traceability.

[P180] is one of the very first works, in our review list, that discusses the influence of cognitive biases in decision making. In this work, Wirfs-Brock indicates the presence of confirmation and information biases in design discussions. Zalewski et al. [P181] conducted a workshop with 14 software engineers to understand the influence of biases in ADM. Based on that workshop, they documented 12 cognitive biases that are prominent in ADM. [P182] also covers the aspect of cognitive biases in ADM and introduces RDM, NDM, and bounded rationality. We agree with the following points made by Tang and Van Vliet in [P182]:

1. People are irrational in general.
2. Rationality of individuals is by what they already know, cognitive limitations of the human mind, and the finite amount of time available to make a decision (referring to [14]).
3. According to the law of least effort, people minimize cognitive load and use intuition in ADM (referring to [71])

These findings align with the Law of Least Efforts – which states that people tend to minimize cognitive load and tend to use intuition in making decisions. Intuitions and cognitive biases lead to unsound design decisions. Antony Tang suggests the following three main reasons of design reasoning failures in [P183]:

- Cognitive bias: relates to “I have a hammer, and everything is a nail”;
- Illogical reasoning: unsound arguments and inferences to reach a design conclusion (basic premises of a decision);
- Low-quality premises: if assumptions are unclear (ambiguous design concerns and requirements), architects could end up making decisions based on faulty inputs;

Weinreich et al. [P184] also argue that “*education, experience, and biases like passion, evangelism, personal preferences of the people in charge, and company values*” greatly influence ADM. Based on the results of twenty-five interviews, they listed down eight factors that influence ADM, namely, company size, business factors, organizational factors, technical factors, cultural factors, individual factors, project factors, and decision scope. With respect to decision scope, they classified ADDs according to granularity, scope, and impact [P185]. The low-impact decisions are typically made by an individual or in a small group, whereas, high-impact decisions are made in larger groups while involving experienced individuals. An important point (for this dissertation) they mention in [P184] is that *architects, in the first place, have to know that there are alternatives otherwise, they do not realize that they are making a decision.*

In [P186], the authors studied the architectural description of open-source projects documented

in [72] and found that quality attributes strongly influence ADM and the decisions in those projects had been made in small teams or by single individuals.

A very interesting and intuitive card game is presented in [P187]. The cards (representing constrain, assumption, risk, and tradeoff) is used to help architects during their ADM process. Since architects can forget to reason about their decisions [P188], such card games could help to spark architects' reasoning process. In our approach by automatically generating and recommending alternatives, we too aim to trigger the thought process of architects during ADM.

The interview results presented in [P189] indicate that the experience of decision maker affects the return of investment of projects. On the other hand, the study also reveals that factors such as the role of a decision maker or if he or she codes the system are irrelevant. With the focus on the source code of the system, the authors of [P190] conducted interviews with 104 software developers and found that they make many decisions concerning the design, scope, and technology use. They indicated that “*developers rarely conceptualize their work as decisions between more than two options and developers have trouble remembering multiple options*”.

An important aspect that should be considered by any ADM support system is that they should *explicitly include reflective questions* for architects to consider during ADM. The authors of [P191] distinguish between two minds: Mind 1 and Mind 2. Mind 1 reflects the design reasoning mind with the problem-solving mindset, whereas, Mind 2 is the reflective mind with a feedback mindset. ADM tools should include *mechanisms to trigger, question, and reflect on the activities performed by Mind 1*.

For a more detailed discussion on the factors influencing ADM, readers are further directed to the work by Tang et al. [P192].

### 2.2.4.1 Optimization-based approaches

For supporting architects during the ADM process, we found twenty-three publications that specifically target to model the ADM process as a constraint satisfaction problem by modeling conflicting criteria in decision making. Since we follow the school of thought that suggests *designing software systems (making ADDs) is a wicked problem*<sup>6</sup>, we believe (based on our observations in industry) that even though, such approaches for ADM are essential, they are not used (or will be used in the near future) by practitioners in industry (especially, for information systems and not embedded systems). Hence, we have not extensively reviewed these twenty-three publications but have tried to summarize the gist of those publications.

To choose the most optimal design alternative that meets a set of criteria, AHP is one of the popularly used approaches. For example in [P193], authors use AHP to select components that meet quality criteria. First, all the goals/objectives are captured hierarchically, the criteria used to achieve those objectives are specified, and the alternative options are listed. Second, at each level of the hierarchy a decision table is created. Finally, a constraint solver prioritizes and selects an optimal alternative that meets the pre-specified constraints.

---

<sup>6</sup>Rittel and Webber referred to ill-defined design problems as wicked problems. These problems are difficult to solve due to contradictory, often incomplete, and frequently changing requirements [73, 74]

The authors of [P194] apply AHP for ADM and base their approach on the following steps:

- Along with cost and benefit values, capture the alternatives in a goal model.
- Capture alternatives' preferences using AHP.
- Use a search-based optimizer to perform heuristic sampling of the decision space.
- Rank the alternatives and select the optimal option.

To select the best alternative that meets different quality criteria [P195] and [P196] also propose to use AHP. Another approach that uses the hierarchical criterion structure based on the criteria importance theory is presented in [P197] which allows the selection of an optimal alternative.

Since architects do not define all the constraints for selecting an optimal solution, authors of [P198] take a different viewpoint on how to apply constraint propagation algorithms to support ADM. Instead of finding the optimal solution, they stress on using constraint solvers for eliminating or reducing infeasible options that are inconsistent with the already made choices.

The design task is represented as a search problem in [P199]. Since ADDs are interrelated, their relationships are captured using two relation types (superior and inferior), and when a superior decision is under consideration, all its related inferior decisions are treated jointly to identify inconsistencies among decisions and to support ADM.

[P200] and [P201] consider ADM to be a Multiple-Criteria Decision-Making (MCDM) problem and represent occurrence probabilities as a first-class entity in ADM. By using occurrence probabilities for each combination of alternatives, the consequence of each alternative can be evaluated. [P202] also uses an MCDM method to select an optimal alternative by making trade-offs between different quality attributes (which are weighted by stakeholders). In [P203] also, the authors apply evolutionary algorithms and MCDM strategies to identify design alternatives.

Fuzzy inference (Choquet integral) is used in [P204,P205] to select the most suitable architectural style that meets stakeholders' different (quantified) goals and objectives. On the use of fuzzy inference, [P206] presents a method called Multicriteria decision aid (MCDA) for the selection of an alternative while considering uncertainty (fuzziness) in stakeholders' requirements. Esfahani et al. [P207] use a fuzzification approach to represent uncertainty as a triangular fuzzy value to support architects explore the solution space under uncertainty.

A web-based tool called Decision buddy is presented in [P208], which allows users to capture issues and alternatives, prioritize alternatives using a constraint solver based on the weights of corresponding quality attributes, and finally approve or reject those alternatives that meet the criteria. Another tool named RADAR which uses a multicriteria optimization approach is presented in [P209]. To select an optimal alternative, they suggest the following steps: (1) model the decision problem and create a decision model; (2) apply montecarlo simulation on the decision model and shortlist the optimal alternative using multicriteria optimization; (3) if required, get more information and repeat steps 1 and 2.

A weighted-score approach using a decision matrix is used to compare choices against decision criteria as well as to compare the options against ideal solutions is presented in [P210]. Imran et al. [P211] also propose a weighted-score approach for ranking and selecting the best architec-

tural pattern that meets the quality goals. Another multiagent-based tool called DesignBots is presented in [P212] which takes as input the initial architecture of the system and a weighted list of quality constraints. Based on the input, the system then suggests different alternatives to improve the architecture.

In [P213], the authors present a model-driven approach for the evaluation of design decisions while considering quality attributes. They claim that their system can be used to select an optimal decision when its impact on the quality attribute is unclear. Another interesting approach that considers ethics in ADM is presented in [P214]. In this work, the authors present a MCDM scheme which considers ethical analysis explicitly.

Finally, the last publication in our review list that focuses on the optimization problem is from Shahbazian et al. [P215]. In this work, the authors present a search and simulation-based approach for understanding the impact of ADDs on the underlying system.

### 2.2.5 Group decision making

We found eleven publications focusing on GDM. They address various topics including the challenges and factors influencing GDM, methods, and models of GDM.

Alali and Sillito [P216] conducted interviews with thirteen architects about their design process and showed that ADM is a group activity. They describe that the motivation for collaboration in response to social commitments and organizational work contexts include: (a) improving ADDs and (b) sharing ADM effort. Another case study [P217] in a large organization shows that the majority of software teams make ADDs collaboratively. In this work, the authors indicate that the majority of the teams prefer consultative decision-making style as it helps to make ADDs while talking into account the considerations of all the team members.

In [P218], the authors present an “ONTOlogy-based Group Decision Support System” which allows architects to participate in the GDM process through a web interface. The system relies on a group argumentation model to support conflict resolution during GDM. Another tool that uses an argumentation viewpoint approach for GDM is the Software Architecture Warehouse (SAW) [P219]. SAW is a collaborative web-based tool that manages AK and also provides features for handling different decision models. In wiki pages, architects can collaborate and discuss architectural choices for design issues, evaluate, and select a suitable option thereof. In [P220], the authors propose a process named GADGET to help architects increase consensus during GDM. GADGET process adheres to the idea of bounded rationality. In this process, for a decision topic, alternatives and concerns are discussed, those alternatives are prioritized, and based on that, architects in a group aim to reach consensus.

In the context of agile projects, the authors of [P221] suggest combining the ADM process with Scrum. They propose four steps of GDM to be included during a sprint planning: (a) problem identification, (b) development of alternatives, (c) preference indication and prioritization, and (d) reaching consensus among group members.

Rekha and Muccini [P222] conducted a survey with thirty participants to gain insights about the challenges in GDM. They found that structured approaches were not used but instead

brainstorming was used in 70% of the companies to reach consensus in GDM. They also found that conflicting decisions and misunderstanding of goals were the challenges in GDM. In another paper [P223], they argue that the current methods do not suit GDM. The methods should include stakeholders' preferences, rules indicating how those preferences should be considered, and mechanisms for conflict resolution. Rekha and Muccini [P224] suggest including GDM strategies into an architecting phase to not only capture ADDs but also to document the GDM factors that result in those ADDs. Based on these observations, they extended the ADD model with concepts from GDM and organizational structures [P225]. This model includes concepts such as groupDecisionSession, groupMembership, stakeholder which are linked to ADDs. In their very recent work, Rekha and Muccini [P226] conducted another survey with 35 practitioners. The results of this survey also show that a standard way of ADM is less common and tools for ADM are rarely used (because the quality of the available ADM tools is below satisfactory). Interestingly, they also indicate that "despite the involvement of team members in discussions, the final decision is made by an individual". Finally, Rekha and Muccini again emphasize the *need for supporting architectural groups by integrating GDM principles into ADM tools.*



The tool named “Amelie - Decision eXplorer (ADeX)” presented in this dissertation is based on the requirements from our industry partner as well as from the challenges identified by researchers as discussed in the previous chapter. Since it was highlighted by many researchers (e.g., [P141]) that the existing AKM tools must consider industry requirements adequately, we made sure that components within ADeX were iteratively implemented after carefully considering the needs of our industry partner.

In 2015, a research project named “Architecture management enabler for large industrial software (Amelie)” was initiated within the Architecture Definition and Management (ADM) research department at a large IT company in Germany. This department comprises of more than forty software architects who provide architectural consulting to various business units. The core idea behind the Amelie initiative was to build an AKM system to ensure that (a) the tacit AK of architects within the department was made explicit, (b) the organization-specific architectural patterns, methods, and experts were systematically documented, and (c) architects receive just-in-time recommendations based on the captured AK during the execution of their projects. Based on this initial idea, two research assistants (including the author of this dissertation) and three architects at the ADM department started the research project. Over the course of four years, eight workshops with six software architects were conducted at regular intervals. The purpose of the workshops was two-fold. First, we wanted to ensure that the findings from the literature align with the expectations/needs of architects, to define the use cases for the system. Second, the implemented prototype (individual component within ADeX) met those expectations and to gather feedback for improvements. Such continuous interactions with the architects as well as the literature review helped us reframe the initial idea of the AKM system. First, the focus shifted to managing ADDs as a first-class AK and supporting architects during ADM in large software projects. Second, instead of a top-down approach to AKM (wherein, based on a model, architects capture ADDs and the tool provides recommendations thereof),

we shifted our focus to a bottom-up approach in which ADDs are automatically retrieved from different information systems and are annotated with publicly available data sources in order to support architects during the ADM process.

Even though the semi-systematic literature review was performed during the later phase of the dissertation to provide a holistic view of ADM, the comparative tool study conducted by Tang et al. [75], an extensive literature review by Capilla et al. [1], the use cases from Liang and Avgeriou [76], and the experiences from Kruchten et al. [77] provided us the comprehensive list of requirements that should be supported by tools for AKM and ADM. These requirements were validated and prioritized by the architects in the earlier mentioned ADM department and are summarized in this chapter. We first present those requirements related to AKM that are broader in scope and then, discuss the requirements specific to ADM and ADD management.

## 3.1 Requirements related to architectural knowledge management

We found twenty-five requirements related to AKM that were mentioned in the literature studies. Out of those requirements, architects in the Architecture Definition and Management department found twenty-two requirements to be highly relevant and three to be partially significant (UC 10, UC 13, and UC 21). Furthermore, due to time constraints, not all the use cases have been implemented in ADeX. However, only sixteen use cases that fit the storyline of our bottom-up approach that focus on supporting architects analyze ADDs have been implemented.

It should be noted that the below-listed requirements in Table 3.1 and Table 3.2 were validated with our industry partner during a Master's thesis project at our research department. These requirements have already been documented by Xu in her Master's thesis report [78].

Table 3.1 shows the list of requirements related to AKM. The first column represents the use case number, the second column captures a short description, the third column shows the reference to the publication discussing the corresponding use case, and finally, the fourth column indicates the relevance to this dissertation. We use 'Y' to indicate if the use case is supported by ADeX and '-' for those use case that are out of scope of this dissertation.

After conducting interviews with three software architects (cf. [78]) who were responsible for the AKM initiative (Amelie), we were not surprised that they considered all the twenty-six use cases identified from literature studies to be relevant. They only suggested that the use cases, namely, versioning AK (UC 10), notifications (UC 13), and translating AK (UC 21) were not of high-priority. Furthermore, many of the below-listed use cases are quite generic and broad which are covered by most of the existing AKM tools. For instance, create, read, update, and delete (CRUD) operations (UC 1) on the AK elements (depending on the AK model) are essential features of any AKM tool. Also, searching (UC 2) persisted information in the knowledge base using search queries or filters, and sharing (UC 3) that information are standard features.



No.	Description	Ref.	ADeX
UC 1.	Managing AK elements and their relationships: create, read, update, and delete AK elements in a knowledge repository.	[79, 80], [P2, P29]	Y
UC 2.	Searching AK: search and view AK elements using search criteria/filters.	[80], [P20]	Y
UC 3.	Sharing AK: share AK with different stakeholders.	[1, 26, 76, 81]	Y
UC 4.	Reusing AK: the architect reuses AK in another project context (e.g., reusing ADDs from an old to a new project and reusing internal or external data sources).	[1, 82], [P20]	Y
UC 5.	Integration with tools: (semi-) automatic integration with tools and different types of information during the SDLC.	[1, 76, 83], [P20]	Y
UC 6.	(Semi-) automatic AK enrichment: generate AK content proactively (e.g., automatically distilling and interpreting AK from text without the users' intervention).	[80]	Y
UC 7.	Comprehending AK: learn and comprehend AK (e.g., understand the rationale of a design decision).	[76], [P20]	Y
UC 8.	Identifying stakeholders: according to certain criteria (e.g., who has the most "weight" on an ADD).	[82], [P2]	Y
UC 9.	Tracing AK: trace between various AK elements (e.g., ADDs and quality attributes).	[84]	Y
UC 10.	Versioning AK: manage different versions of the AK.	[85], [P2]	Y
UC 11.	Collaborative environment support: concurrent access for multiple users.	[P8, P20, P29]	Y
UC 12.	Customizing and configuring AK: a comprehensive and tailorable AK representation to support and evolve AK. Provide organization-, domain-, or project-specific AK models and tools. Provide users with different levels of permissions.	[26], [P20, P29]	Y
UC 13.	Notification about AK changes: subscribe to specific AK elements, and get notified about changes to them.	[P9]	Y
UC 14.	Distilling AK: distill AK from a system into general knowledge (e.g., architectural styles) that can be reused.	[P20]	Y
UC 15.	Applying general AK: use application-independent/static AK (e.g., architectural styles to solve the problems at hand).	[76], [P20]	Y
UC 16.	Synthesis and automated decision-making support: provide automated support for architects during AKM (e.g., recommend experts for a certain decision or provide multiple potential solutions to address a concern).	[80, 86], [P9, P20]	Y
UC 17.	Maintenance support: analysis of changes on AK elements.	[86, 87], [P21]	-
UC 18.	Implementation support: trace the implementation artifacts to ADDs.	[44, 88], [P20]	-
UC 19.	Architectural evaluation support: analyze trade-offs and risks of each solution. Perform a critical evaluation of the AK, e.g. to make sure that requirements have been satisfied in the architecture design.	[82, 89], [P20, P29]	-
UC 20.	Reasoning capability: basic inference mechanism that can automatically detect inconsistencies, state new questions or perform some fuzzy reasoning.	[P29]	-
UC 21.	Translating AK: facilitate reuse by translating the formal AK based on a domain model into another domain model.	[90]	-

### 3. Requirements elicitation

---

UC 22.	Support trade-off analysis: analyze the architecture by trading off different quality attributes.	[82], [P20]	-
UC 23.	Cleaning up architecture: users make sure that all the dependencies of removed AK (e.g., the consequences of an ADD) have been removed as well.	[82]	-
UC 24.	Support design maturity assessment: the architect evaluates when the architecture can be considered as finished, complete, and consistent (e.g., verify whether a system conforming to the architecture can be made or bought).	[82]	-
UC 25.	Risk analysis: identify flaws in the architecture.	[82]	-

---

Table 3.1: AKM related use cases, their relevance to this dissertation, and their sources

With respect to UC 4, reuse of AK depends on the purpose of an AKM system. For example, in ADeX, reuse can be viewed from two perspectives. One, by highlighting the design decisions made in large software systems or by showing similar design decisions made in the past, we hope that architects and developers are implicitly able to apply them in similar context. Second, we achieve reuse by explicitly using AK maintained in public data sources like the DBpedia ontology to recommend alternative solutions for addressing design concerns. The same goes for UC 7, comprehending AK depends on what information is presented to the users of the system. In ADeX, users can reason about how many decisions were made to address a specific quality concern, which architectural elements are affected by those design decisions, or who should be involved in making a design decision (UC 8). Reasoning about related AK elements also links to the tracing AK use case (UC 9). Those indications about which elements are affected by a design decision can only be achieved if the system provides mechanisms to capture such relationships.

AK can be broadly categorized into general and application-specific AK. The general AK (UC 14) (e.g., architectural styles, design patterns) which often does not change is applied in an evolving project-specific context (UC 15). In our system, we distinguish between these two types of AK categories and represent them using different domain models.

Some use cases such as versioning AK elements (UC 10) and notifications about AK changes (UC 13) are facilitated as out-of-the-box features if we use any wiki-based systems (e.g., MediaWiki and SocioCortex) as the backend of the AKM system. Furthermore, use cases in the context of multi-user support, namely, collaborative environment (UC 11) and customizing and configuring AK (UC 12) are de-facto features in today’s web-based systems. Any production-ready system should provide features that allow multiple users to tailor the views that best fit their project needs. However, it should be noted that most of the existing systems only support configuration options for the user interfaces, but as highlighted in [P29] the domain models behind the AKM tools must also be configurable to meet the organization-, domain-, and project-specific needs. We specifically address this use case with the use of a meta-model based system that allows users to configure the domain models at runtime (discussed later in Chapter 5).

Many researchers have already highlighted the need for AKM tools to integrate with other systems regularly used by architects and developers. We provide particular emphasis on this use case (UC 5) to ensure that ADeX can automatically import and synchronize data from systems like MS Project, JIRA, Github issues, and Enterprise Architect. Furthermore, we not only use

data from organizations' internal systems but also publicly available data sources to enrich AK automatically (UC 6) and to generate recommendations (UC 16).

As elaborated in the later chapters, the first sixteen use cases are currently supported by ADeX, and the use cases UC 17 to UC 25 are considered to be out-of-scope for this dissertation. The rationale for not including them in our scope includes not only the time and effort constraints but also the fact that we focus on the analysis of natural language text extracted from sources such as issue management systems to provide recommendation support. Hence, use cases such as translating AK (UC 21), design maturity assessment (UC 24), or risk analysis (UC 25) does not fit the context.

## 3.2 Requirements related to architectural decision making

In the previous section, we presented those use cases that were in the context of AKM. In this section, we list eighteen use cases that address ADM. These use cases have also been extracted from the literature review discussed in Chapter 2. Since ADDs are an integral part of AK, some of the generic use cases such as creating and updating ADDs have already been covered in the previous section. We follow the same scheme as in the previous section to represent the priorities of the use cases in Table 3.2 and discuss those use cases that are relevant for this dissertation. 'Y' in the fourth column indicates that the use case is supported by ADeX and '-' denotes that the use case is out of the scope of this dissertation (for the same reasons as discussed in the previous section).

It should be noted that when the Amelie research project was initiated in 2015 by our industry partner, the focus was merely on AKM. However, over time, by incorporating the trends in the software architecture research community, the focus slightly shifted towards the management of ADDs within the AKM system.

Table 3.2 summaries the use cases that are specific to the management of ADDs. The challenge of interoperability of the AKM systems with those systems regularly used by architects during the SDLC has already been discussed in the previous section. Similarly, the authors of [91] also discuss the need to support the import of ADDs from different systems. In this dissertation, we enable the extraction of projects' data from different systems (UC 26) such as MS Project, JIRA, Github Issues, and Enterprise Architect. Once the information is extracted, it is preprocessed, and design decisions already made and implicitly captured in those tools are extracted using a ML algorithm (UC 27). Once design decisions are automatically extracted from projects' data, the subsequent preprocessing step annotates the descriptions of those decisions using NLP techniques. The NLP techniques help to identify and link architectural elements and quality attributes addressed by the extracted design decisions. The links or the relationships between design decisions and its associated concepts such as quality attributes and architectural elements are modeled in the knowledge base using a meta-model based approach.

### 3. Requirements elicitation

No.	Description	Ref.	ADeX
UC 26.	Support ADD import and export: allow information exchange between different systems	[91]	Y
UC 27.	ADD recovery: architects reconstruct decisions with their associated rationale from an existing or 3rd party system.	[82]	Y
UC 28.	Support complex representation of ADDs: use modles/ontologies to represent the complex nature of ADDs, as well as their dense inter-dependencies	[P20,P29], [82]	Y
UC 29.	Searching and filtering ADDs: by role, phase, and scope	[90]	Y
UC 30.	Retrieving ADDs: given the architectural model, trace back to the ADD it is based on. Provide the drivers and the rationale of the decisions	[44,82,90,92]	Y
UC 31.	ADD visualization: depending on the stakeholders' concerns visualize ADDs and their relations using different viewpoints	[82,93]	Y
UC 32.	Identifying architectural driver: identify which architectural drivers have the most influence on the design of the system	[82]	Y
UC 33.	ADD consequence analysis: the main consequences of a decision are the changes in the model when a decision is executed. In addition, new decision topics can be introduced	[44,82,92]	Y
UC 34.	Support stakeholder-specific overview: identify key ADDs and unresolved concerns for a specific stakeholder	[82]	Y
UC 35.	Recommend experts: who should be involved in ADM	-	Y
UC 36.	Similar ADDs: for making a new ADD, find similar already made ADDs	-	Y
UC 37.	Recommend alternative choices: while making new ADDs	-	Y
UC 38.	Affected stakeholders identification: identify which stakeholders will be affected by changes in decisions	[82]	-
UC 39.	Pattern detection of ADD dependencies: identify patterns in the graphs of decisions that can be interpreted in a useful fashion. E.g., decisions being hubs ("Godlike" decisions), circularity of a set of decisions, and decisions that gain weight over time and are thus more difficult to change or remove	[82,92]	-
UC 40.	Check implementation against ADDs: check how is the implementation is in line with the ADDs, to know where in the development process people disregard the made decisions	[92]	-
UC 41.	Completeness check: check if the requirements are all sufficiently covered by the decisions	[82,92]	-
UC 42.	Consistency check: check if the current ADDs is internally consistent. Check if the chosen alternatives have inconsistent consequences on the architectural model	[82,92,94]	-
UC 43.	Superfiuous ADD detection: check decisions that overlap (e.g., decisions do not affect the current architectural model at all and are unnecessary)	[82,92]	-

Table 3.2: ADM related use cases, their relevance to this dissertation, and their sources

The automatically extracted design decisions and the corresponding annotated architectural knowledge are persisted in the knowledge base<sup>1</sup> that maintains the inter-dependencies (UC 28). The system then allows end-users to filter and search design decisions (UC 29) from the knowledge base that affect specific quality attributes and architectural elements attributes. UC 30 is related to the previous use cases of linking the design decisions to their associated concepts such as architectural elements and quality attributes. These concepts form the drivers and the rationale of the made design decisions. The design decisions and the related concepts are presented to the end-users using different views comprising of different visualizations (UC 31) in the ADeX system. For instance, the quality attributes influencing the design decisions are presented using a bar chart visualization and the architectural elements affected by those design decisions are shown in a bubble chart (cf. Chapter 5 for an in-depth discussion). Furthermore, these visualizations, also help to indicate those quality attributes and architectural elements that have the most or the least influence on the design decisions (UC 32). When a new design decision is made, it can introduce new architectural elements (UC 33). This use case is reflected in the architectural elements visualization through the introduction of new nodes representing those elements. Another visualization uses a tabular representation to display all the already made design decisions along with its related concepts including the status, priority, and the assigned individual to those decisions (UC 34).

We introduced the next three use cases based on the inputs from our industry partner and their relevance given the context of ADM. The use case (UC 35) supports architects by recommending experts who could be involved during the ADM process. While making new design decisions, if similar decisions have been made in the past, architects get recommendations about them (UC 36). Such a use case helps architects explore those previously made similar decisions and understand the impact of such a decision from the perspective of their impact, effort, and cost of realizing such design decisions. The third use case in this list (UC 37) is focused on the process of ADM. That is, when making design decisions, architects have to consider and evaluate multiple alternatives to choose an alternative that meets the requirements. To aid this process, ADeX automatically generates these alternatives along with the meta-data from a publicly available data source and presents them to the architects. The purpose of such a use case is to trigger the thought process of architects to consider or at least be aware of the possible alternatives while making design decisions.

The use cases (UC 38-43), even though relevant in the context of ADM, have not been covered in this dissertation. Some of these use cases are discussed in Chapter 7 as part of our future work. For instance, UC 40 is a logical extension to our work. Currently, we only link design decisions to issues in task management system; these issues can be further linked to code commits in version control systems and hence, establish traceability from decisions to their implementation.

In total, 16 use cases specific to AKM and 12 use cases related to ADM are supported by the system (ADeX) presented in this dissertation. It should be noted that, as already mentioned, there are few use cases that overlap (e.g., UC 5 and UC 26). Such scenarios are highlighted in the subsequent chapters accordingly. The next chapter elaborates on the architecture of the ADeX and highlights how ADeX supports the aforementioned use cases.

---

<sup>1</sup>The ML algorithm, NLP techniques, and meta-model based approach are elaborated in the next chapter



---

### A conceptual framework for architectural decision making

---

The conceptual AKM framework shown in Figure 4.1 addresses the ADM related challenges discussed in the previous chapter. In this chapter, we briefly present the components within the framework and link them to the use cases presented in Table 3.1 and Table 3.2. The AKM framework integrates within software architects' and developers' working environment to:

- consolidate projects' data from disparate data sources
- extract design decisions made in legacy software systems
- highlight and reason about design decisions' rationale
- recommend alternatives that could be considered while making new design decisions
- find experts who should be involved in decision making
- recommend similar decisions to estimate the time and effort required for addressing new design concerns

This chapter presents how these use cases are addressed by realizing the independent components within the proposed framework. We propose to track the current state of a project, extract information from project artifacts, and support software architects and developers to address specific architectural concerns. This high-level objective is accomplished through the following tasks which are handled by different components within the framework:

- Develop an evolvable and configurable domain model to capture project-specific information including project context, requirements, ADDs, tasks, and stakeholders in a meta-model based AKM system.
- Extract, Transform, and Load (ETL) project-related data from different tools (such as Excel, MS Project, Enterprise architect, and JIRA) into the AKM system.

#### 4. A conceptual framework for architectural decision making

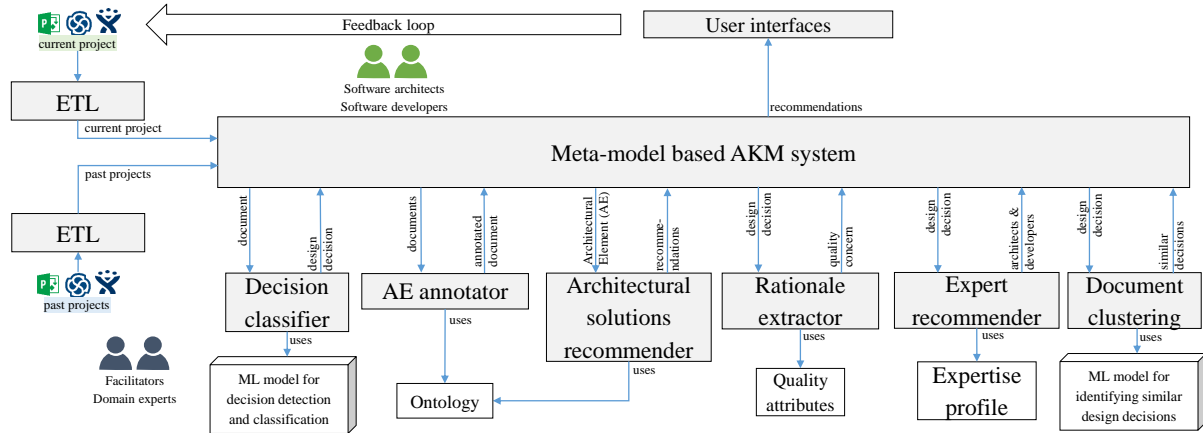


Figure 4.1: A conceptual framework for managing ADDs and supporting the ADM process

- Implement an approach to extract design decisions made in the past. Focus on automatically detecting, extracting, and classifying design decisions from issues maintained in issue management systems such as JIRA and Github issues.
- Device an approach to automatically identify architectural elements in the text.
- Populate the knowledge base of the AKM system with alternative architectural solutions. Given an architectural concern, the system should recommend architects and developers to consider available alternative solutions.
- To determine the rationale behind design decisions, link them with quality attributes.
- Design a user-profile model to capture decision makers' preferences. User preferences, in the context of ADDs, for instance, include (a) frequently used technologies, plugins, and libraries, (b) inclination to address concerns related to particular architectural elements or concerns related to specific quality attributes of the system. Use such a user-profile to recommend relevant experts to discuss new design concerns.
- Develop an approach to detect similar design decisions made in the past.
- Using a web-based User Interface (UI), present different perspectives and recommendations to the end-users of the system.

### 4.1 Stakeholders of the framework

As shown in Figure 4.1, there are two main roles within the proposed framework; stakeholders who are the end-users of the system and those stakeholders who ensure that the necessary information is correctly maintained in the AKM system or facilitate the end-users of the system.

**End-users:** Software architects (solution and system architects) and software developers are the primary end-users of the system. However, in specific scenarios, the realized framework is also beneficial to project managers. End-users have the flexibility to import their projects



using the ETL platform. After the automatic transformation of the imported project's data by different components, end-users are presented with different perspectives that aid them to understand design decisions made in the past. Different perspectives allow software architects and developers to narrow down on those aspects that are relevant to their given context. For instance, in a specific project, if a software architect is concerned with the performance of the respective system and wants to explore and learn from decisions made in the past, a specific UI provides the possibility to that architect to analyze only those design decisions that were made to address performance related concerns. After a retrospective analysis, end-users can make new design decisions and update the necessary artifacts within their project as part of the feedback loop. One of the core benefits of this framework is that it does not intrude, or force end-users to actively maintain the AKM system, or use any specific tool. They can continue to use the information systems such as Jira, MS Project, or Enterprise architect to document their discussions in their projects. Hence, as shown in Figure 4.1, the feedback loop links back to these aforementioned systems. The new information, henceforth, is brought back into the AKM system using the ETL component.

**Facilitators:** The second group of stakeholders who are the facilitators of the system is shown in the lower part of Figure 4.1. These facilitators are either the maintainers of the system or domain experts who ensure that the necessary information is correctly maintained in the AKM system. The maintainers of the system are software developers who, for instance, provide the correct mapping between the ETL component and the AKM system, implement new connectors for importing data from a new information system, or fix any bugs and maintain the system over time. On the other hand, for industry-specific projects, domain experts related to those projects play a crucial role. For instance, corresponding to the architectural element component or the expert recommender component, domain experts provide input for including organization- and project-specific architectural elements that, for example, are not available in the lookup ontology shown in Figure 4.1.

The responsibilities of these two categories of stakeholders compliant each other. If the end-users of the system notice discrepancies in the recommendations, they can also actively update the recommendations through the User Interfaces (UIs), but can also communicate those issues to the domain experts and maintainers of the system. On the other hand, after an iteration of analysis of a given project, maintainers of the system can request for feedback from the end-users to improve components' services for subsequent iterations.

## 4.2 Components and their responsibilities within the framework

Figure 4.2 shows the realization of the AKM framework presented in the previous section. The realization of the framework in collaboration with our industry partner is named ADeX. ADeX is the acronym for "Amelie - Decision eXplorer" where Amelie stands for "Architectural Management Enabler for Leading Industrial software". The subsections briefly present each of the components within ADeX. The detailed description of each of the component is presented in the next chapter.

#### 4. A conceptual framework for architectural decision making

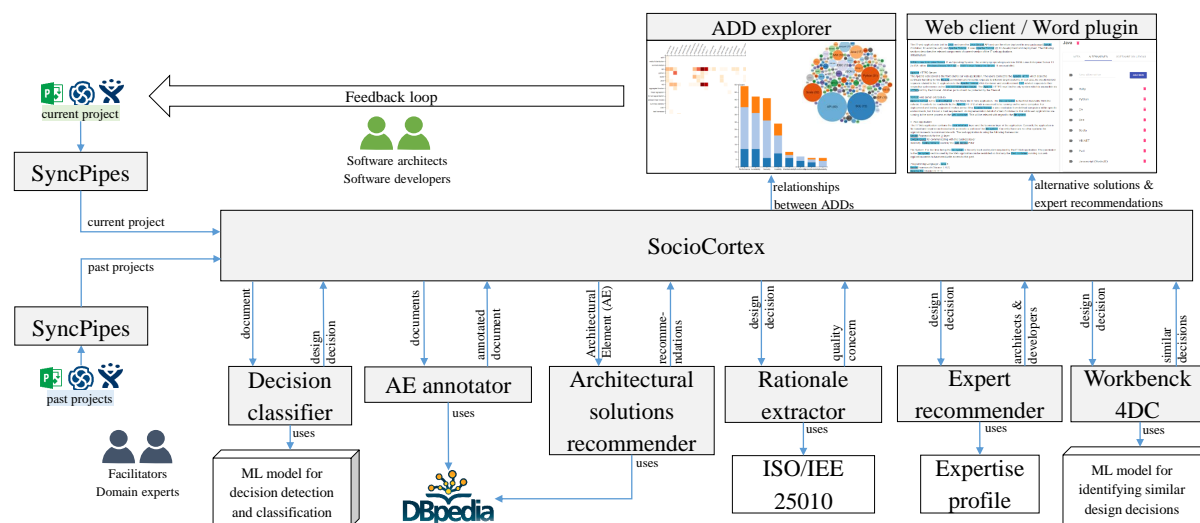


Figure 4.2: The realization of a conceptual framework for managing ADDs and supporting ADM

##### 4.2.1 SocioCortex: A meta-model based AKM system

SocioCortex is a collaborative knowledge management system which follows the hybrid-wiki approach [95]. The hybrid-wiki approach handles scenarios wherein a system’s domain model as well as the data captured by the system evolve simultaneously. Notably, in the early phases of the development of a new software, new requirements emerge and the understanding of the domain evolves. Hence, in such situations, the domain model of the system needs to evolve constantly.

Furthermore, in the context of an AKM system, stakeholders in different projects use different terminology to refer to the same or similar concepts. For instance, if in one project, the term *Issue* is used, then in another project, *Task* could be used to represent the same concept. Moreover, the manner in which relationships are maintained between concepts in the domain model may also vary. For instance, if in one project, stakeholders link tasks to quality requirements then in another project this relationship may be nonexistent, or tasks could be related to a general concept of requirement. In such scenarios as well, the domain model of the underlying software system needs to be adapted accordingly.

A model-based approach or the idea of **models@runtime** facilitates stakeholders of different projects to configure their domain models so as to meet the system requirements. The meta-model of such an AKM system and the following use cases supported by SocioCortex (out-of-the-box features) are further elaborated in Chapter 5.

**UC 1:** Managing AK elements and their relationships: create, read, update, and delete AK elements in a knowledge repository.

**UC 2:** Searching AK: search and view AK elements using search criteria/filters.

**UC 3:** Sharing AK: share AK with different stakeholders.

**UC 4:** Reusing AK: the architect reuses AK in another project context (e.g., reusing ADDs from an old to a new project and reusing internal or external data sources). [Clone functionality]

**UC 9:** Tracing AK: trace between various AK elements (e.g., ADDs and quality attributes). [Using relationships between concepts in the AK model]

**UC 10:** Versioning AK: manage different versions of the AK.

**UC 12:** Customizing and configuring AK: a comprehensive and tailorable AK representation to support and evolve AK. Provide organization-, domain-, or project-specific AK models and tools. Provide users with different levels of permissions.

**UC 13:** Notification about AK changes: subscribe to specific AK elements, and get notified about changes to them. [Watch a wiki-page feature]

**UC 14:** Distilling AK: distill AK from a system into general knowledge (e.g., architectural styles) that can be reused. [Static and dynamic knowledge models]

**UC 28:** Support complex representation of ADDs: use modles/ontologies to represent the complex nature of ADDs, as well as their dense inter-dependencies.

At a high level, SocioCortex can be seen as a backend-as-a-service (BAAS) system that allows to model and capture AK related to a project. Once the project related concepts such as Project, Issue, Stakeholder, Quality attribute, etc. are modeled, data corresponding to each of those concepts within a project can be captured in the system. The other components within ADeX use the APIs provided by SocioCortex to read and write data for fulfilling their responsibilities.

### 4.2.2 SyncPipes: A data integration and synchronization platform

The need for integrating data from different information systems such as a project management, issue management, and architecture modeling system has already been emphasized in the past [1, 75, 76, 83]. The SyncPipes component addresses that specific need:

**UC 5:** Integration with tools: (semi-) automatic integration with tools and different types of information during the SDLC.

**UC 26:** Support ADD import and export: information exchange between different systems.

SyncPipes follows a model-based approach which allows developers to implement adapters within the platform that are not only reusable but also configurable at runtime. SyncPipes abstracts the tasks involved in importing data from external sources into the target system. Developers only need to implement an extractor and a loader. The extractor specifies how to retrieve data from source systems like MS Project, Jira, or Enterprise Architect. On the other hand, a loader corresponds to the logic of importing data into the target system. In the given scenario, the target system is SocioCortex. The target system can also be any other classical database such as MySQL, MongoDB, or Neo4j. Once extractors and loaders are implemented, stakeholders have the flexibility to choose a specific extractor and a loader using a web-based UI. Furthermore, they can also specify the mapping between concepts in the source system and the target system

for extracting and loading the data. For example, end-users can map the concept Issue in Jira to the concept Task in the AKM system through a UI.

Once the aforementioned mapping is specified between the source and the target system, users can specify how frequently the data needs to be imported and synchronized from the source system into the target system. Finally, using the configurations of the source and the target system as well as the mapping configurations, users can start the pipeline for importing the data. In the current implementation, extractors have been implemented for Excel, MS Project, Jira, Github Issue tracker, and Enterprise Architect. And the loaders include SocioCortex and Mongo database. The technical details about the SyncPipes platform are elaborated in Section 5.2.

### 4.2.3 Decision classifier: A machine-learning based document classifier

As discussed in the related-work section, design decisions are not explicitly documented in large software projects. However, those design decisions are implicitly captured in different information systems. These systems could include issue management systems, chat applications, wiki systems, and e-mail clients. To detect decisions from discussions maintained in the such systems, a decision classifier component was integrated into the framework.

Once all the issues have been imported using the previously discussed SyncPipes component, the decision classifier component identifies those issues which correspond to design decisions. That is, an issue is labeled as either a “design decision” or “not a design decision” (binary classification). If an issue is marked as a design decision, then it is further classified into one of the three decision categories, namely, structural, behavioral, and non-existence/ban design decisions. Further details about the classification approach to detect design decisions are presented in Section 5.3.

The primary responsibility of this component is to identify those discussions that relate to design decisions. Let us consider, a large software project which has twenty thousand issues. This component helps to find those critical issues (e.g., in the range of 500 to 1500 issues) that represent design decisions which impact the design of the system.

### 4.2.4 Architectural elements annotator: A named-entity extractor

This component addresses the challenge of identifying architectural relevant topics within natural language text. Once an issue has been imported into SocioCortex and the decision classifier has labeled those issues (as explained in the previous section), the textual description of all those issues that are labeled as design decisions is annotated with architectural elements. The textual description comprises of information maintained in the title and the description of an issue.

The annotated architectural elements could include terms related to architectural styles, patterns, software technologies, communication protocols, or data formats. These architectural elements are identified using concepts captured in a publicly available cross-domain ontology named DBpedia<sup>1</sup>. The textual description of design decisions is matched with the concepts in the DBpedia ontology, and those concepts that occur in the textual description are tagged

---

<sup>1</sup><http://dbpedia.org>

and persisted in SocioCortex. These annotations linked to the textual descriptions are further highlighted in a UI. End-users can click on those annotations to learn about architectural elements and to get recommendations about alternative architectural elements. By automatically identifying architectural elements in design decisions, this annotator component addresses:

**UC 6:** (Semi-) automatic AK enrichment: generate AK content proactively (e.g., automatically distilling and interpreting AK from text without the users' intervention).

**UC 17:** Implementation support: trace the implementation artifacts to ADDs.

**UC 32:** Identifying architectural driver: identify which architectural drivers have the most influence on the design of the system.

### 4.2.5 Architectural solutions recommender: An ontology-based approach

Due to the fast changing technological landscape for software development, it is practically infeasible for architects and developers to be up-to-date with new and emerging technologies. Furthermore, during the ADM process, architects and developers need to evaluate “some” of the available alternative solutions before making a decision. Hence, to facilitate architects and developers during the decision-making process, architectural solutions recommender component was added to the AKM framework. This component supports the following use cases:

**UC 7:** Comprehending AK: learn and comprehend AK (e.g., implications of an ADD).

**UC 18:** Synthesis and automated decision-making support: provide automated support for architects during ADM.

**UC 37:** Recommend alternative choices: while making new ADDs.

The architectural solutions recommender provides two main services. First, when an end-user selects an annotation within the textual description of a design decision (identified according to the discussion in the previous section), the meta-information about the selected architectural element is presented in the UI. The meta-information of an architectural element includes the element's description, its license (open- or closed-source), underlying technology, and links to the corresponding Wikipedia article. The meta-information of the annotated architectural elements is retrieved from the respective attributes of concepts in the DBpedia ontology. The second service provided by this component includes two types of recommendations:

**Rec I** is the recommendation of software solutions to support an ADD. As discussed before, the variety of software solutions to support an ADD is not always available to software architects. Guiding an architect during the design phase through recommendations of possible software solutions might improve the possibility that an architect will consider multiple alternatives before making an ADD. For instance, if an architect realizes, that under project constraints most of the available software solutions to implement an ADD are proprietary, then she can re-negotiate with stakeholders before making the decision.

**Rec II** is the recommendation of alternative solutions (architectural styles, patterns, and technologies) related to an ADD. Contrary to the previous type, where the aim is to recommend software solutions for higher-level concepts, here, alternative solutions belonging to the same ab-

straction level are proposed. During architectural documentation, providing relevant alternatives allows architects to reason about their ADDs in consideration with the available alternatives.

The main idea behind providing the aforementioned recommendations is that these suggestions will act as a stimulus (trigger) that might lead architects and developers to think about alternative options before making a design decision. The algorithms to generate these recommendations using the DBpedia ontology is elaborated in Section 5.5.1.

### 4.2.6 Rationale extractor: Identifying the rationale behind design decisions

The next component in the pipeline, the rationale extractor component, extracts architectural knowledge from natural language text. As discussed in Section 2.2.2 (cf. Figure 2.8), a design decision is made to address stakeholders' concerns. These concerns correspond to the quality<sup>2</sup> requirements of the software system. Since design decisions are made to address quality concerns, they in-turn reflect the rationale behind those decisions. The design rationale forms the core of the ADM process [43]. Given that design decisions are not explicitly documented in software architecture documents, understandably, the rationale behind design decisions affecting the design of the system is also missing thereof.

Once design decisions are extracted using the decision detection ML model, the rationale extractor component extracts the possible rationale behind the identified design decisions by associating quality attributes to those design decisions.

The quality attributes defined in the ISO/IEC 25010 standard are used as a reference during the extraction process. The quality attributes described in the ISO/IEC 25010 quality model encompass all the quality attributes that are part of the FURPS model. FURPS stands for Functionality, Usability, Reliability, Performance, and Supportability. The FURPS model [96,97] that was initially developed at Hewlett-Packard is widely used in software industry. Similar to the ISO/IEC 25010 quality model, the quality attributes are hierarchical classified in the FURPS model. For instance, performance of a software system deals with speed, throughput, efficiency, resource consumption (keywords: power, ram, cache, etc.), and capacity. The quality attributes, their subcategories, as well as their associated keywords are used to automatically associate design decisions with the respective quality attributes. The automatic extraction process to identify the rationale and to link them to design decisions addresses the following use cases. Further details about the extraction process that uses a keyword-based (indicator term based) approach are provided in Section 5.5.2.

**UC 28:** ADD recovery: architects reconstruct decisions with their associated rationale from an existing or 3rd party system.

**UC 30:** Retrieving ADDs: given the architectural model, trace back to the ADD it is based on. Provide the drivers and the rationale of the decisions.

---

<sup>2</sup>ISO/IEC 25010 and FURPS model include functionality as part of quality attributes.

#### 4.2.7 Expert recommender: User-profile based recommendations

The expert recommender component provides end-users suggestions about architects and developers (experts) who could be involved in the ADM process for addressing new design concerns. In particular, the following use cases are addressed by this component:

**UC 8:** Identifying stakeholders: according to a certain criteria.

**UC 16:** Synthesis and automated decision-making support: provide automated support for architects during ADM (e.g., recommend experts for a certain decision or provide multiple potential solutions to address a concern).

**UC 35:** Recommend experts: who should be involved in ADM.

The expert recommender component uses the results from the previously discussed components to model the architectural expertise of architects and developers within a software project. That is, once the architectural elements and quality attributes are identified using the architectural elements annotator and the rationale extractor component for all the design decisions within a project, those architectural elements and quality attributes are used to create a so-called expertise matrix. And to quantify the architectural expertise of architects and developers within a project, their experience related to addressing design concerns for specific architectural elements is used. If an architect or a developer has addressed design concerns associated with an architectural element, then that individual is considered to have expertise on that topic.

Within the expertise matrix, rows represent software architects and developers within a project and columns correspond to all the identified architectural elements in that project. The cells in the expertise matrix contain integer values which indicate the expertise of an individual on a specific topic. The textual description of a new design concern is matched with the expertise matrix to find suitable architects and developers who have expertise in resolving that design concern. From the list of recommended experts, the end-user can filter and select those experts who need to be involved in addressing the design concern. It should be noted that, the expertise matrix based approach assumes that architects and developers tend to address those issues that they are familiar with and have expertise in. The expertise matrix approach which is elaborated in Section 5.5.3 and has been evaluated using two open-source projects and two industry projects as discussed in the evaluation Section 6.1.3.

#### 4.2.8 Workbench4DC: Clustering similar design decisions

The last component in the AKM framework for extracting AK is the document clustering (Workbench4DC) component. This component addresses – **UC 36:** Similar ADDs: for making a new ADD, find already made similar ADDs.

Workbench4DC component provides various configuration options for the facilitators of the ADeX system to tune and generate cluster models of documents. In the given scenario, these documents correspond to design decisions. Once all the design decisions are identified using the decision classifier component, the pipeline for generating clusters of similar decisions is executed by the Workbench4DC component. Facilitators of the system can specify the classification

algorithm (for example, k-means) and configure the preprocessing steps as well as the parameters (for example, the value of k or the number of cross-validations). Using the clustering algorithm, for a given project, a cluster model of similar design decisions is created and persisted in the file system. For a new and open design concern, its description is compared against the cluster model to identify the cluster to which it belongs. Then, the description is compared against each of the design decision within that cluster to find the most similar design decisions using cosine and Jaccard similarity score. Details about the classification algorithm and the similarity scores are elaborated in Section 5.4. These similar design decisions are finally presented to the end-users using the UIs of the ADeX web-client. Thus, allowing end-users to quickly look up similar design decisions made in the past to understand their complexity as well as to estimate the time and effort required to address similar design concerns.

### 4.2.9 Amelie - Decision Explorer: User interfaces for end-users

The results of all the preprocessing steps for extracting AK from a software project are consolidated and presented to the end-users of ADeX through the UIs in a web client. Using the decision explorer client, end-users can investigate their projects by importing, for instance, all the respective issues from Jira into ADeX. The functionality of all the previously discussed components are automatically triggered and does not require any involvement of the end-users. Once all the issues are processed and AK is extracted, end-users can navigate through various visualizations to analyze their projects. For example, a UI with a bar chart shows all the quality attributes addressed by different design decisions made in the project or the UI with the matrix shows the expertise of architects and developers on specific architectural topics. All the UIs provided by the ADeX client is discussed in detail in Section 5.6. Though those UIs, the following use cases are fulfilled:

**UC 11:** Collaborative environment support: concurrent access for multiple users.

**UC 29:** Searching and filtering ADDs: by role, phase, and scope.

**UC 31:** ADD visualization: depending on the stakeholders' concerns visualize ADDs and their relations using different viewpoints.

**UC 33:** ADD consequence analysis: the main consequences of a decision are the changes in the model when a decision is executed. In addition, new decision topics can be introduced.

**UC 34:** Support stakeholder-specific overview: identify key ADDs and unresolved concerns for a specific stakeholder.

Using the UIs in the ADeX client, end-users can focus only on those design decisions that are relevant for them in their given context and save considerable amount time which otherwise would have been required in searching, for example, a large dataset of issues maintained in their projects. Furthermore, some of the UIs provide necessary recommendations that help end-users make informed and sustainable decisions by exploring similar decisions made in the past.



### 4.3 Process steps within the AKM framework

This section summarizes the steps that an end-user typically follows while using ADeX. ADeX supports software architects and developers to perform retrospective analysis of design decisions in legacy software projects from different perspectives and aids them to make informed future decisions based on those perspectives.

For discussing the steps that an end-user follows within ADeX, let us consider the Apache Spark project. The contributors of Apache Spark project have maintained 21,660 issues in Jira over the last six years. Since design decisions are not explicitly documented within those issues and architects and developers (especially, newly joined contributors) would not know which design decisions were made in the past or to whom to ask about new design concerns. ADeX helps them to address those concerns by automatically extracting AK in textual description and providing different visualizations for addressing their concerns.

First, all the 21,660 issues from Apache Spark's Jira are imported into ADeX's knowledge base (SocioCortex) using the SyncPipes component. Then, the decision detector and classifier component labels those issues that represent design decisions. As shown in Figure 4.3, out of 21,600 issues, 466 issues are labeled as design decisions. Out of these 466 design decisions, 226 design decisions are further classified as structural decisions, 389 and 166 design decisions are tagged as behavioral and non-existence/ban design decisions.

Next, in the augment step, different components further process the textual description of each of the extracted design decision. For example, the architectural elements annotator component annotates all the architectural elements within the textual description of design decisions and also generates alternative architectural solutions as recommendations and persists them in ADeX's knowledge base. These annotated architectural elements are not only highlighted in the textual description within the web UI but are also used to create the visualization of design decisions concerning specific architectural elements. This visualization is represented as a bubble chart as shown in Figure 4.3.

After completing the step of annotating architectural elements, ADeX automatically starts to identify the rationale behind design decisions using the rationale extractor component. This step, highlights, and links all the extracted design decisions with the quality attributes of software systems to indicate why a design decision was made. These links are also used to create a different perspective for the end-users. End-users using the stacked bar chart, as shown in Figure 4.3, can focus only on those design decisions addressing specific quality attributes. For example, if a newly joined architect or developer in the Apache Spark project wants to investigate which design decisions were already made to address performance related concerns, she can click on the corresponding bar in the visualization to examine only those design decisions. For instance, out of 466 design decisions in the Apache Spark project, 54 design decisions were made to address performance related concerns; out of which, 12 design decisions were related to structural design decisions, 28 corresponds to behavioral, 14 design decisions represent non-existence/ban design decisions. This way, for instance, by viewing only 12 design decisions (out of 21,600 issues), end-users can quickly investigate what structural decisions were made in the past to improve the performance of Apache Spark.

#### 4. A conceptual framework for architectural decision making

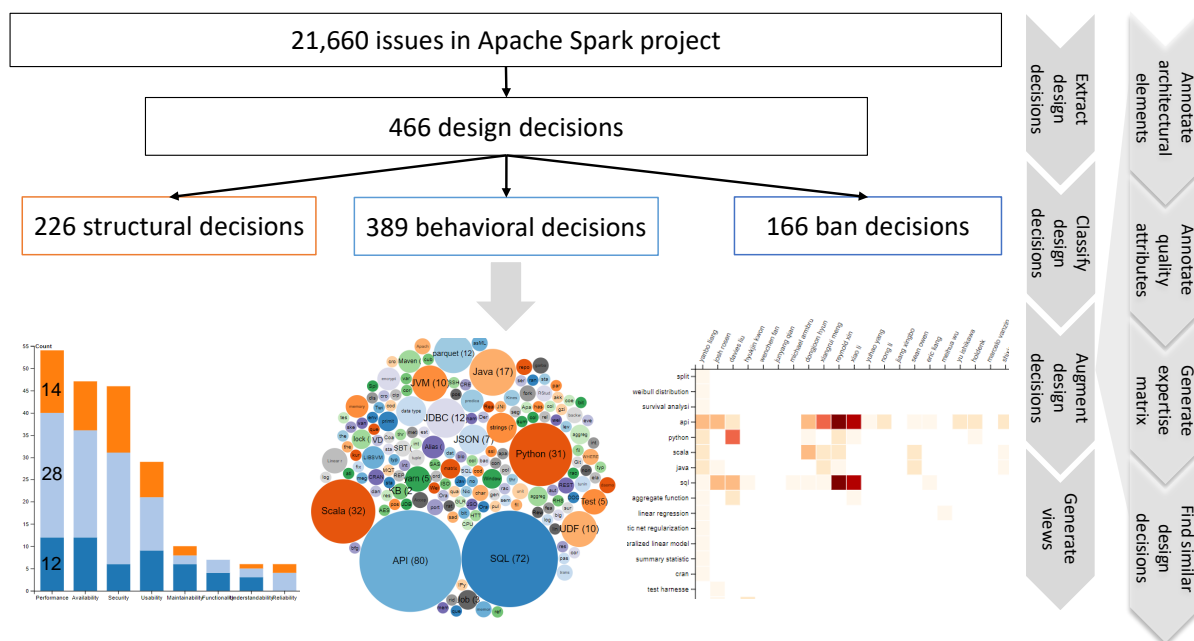


Figure 4.3: The process steps within ADeX to support the ADM process: using the Apache Spark project as an example scenario

The next step uses the results from the previous steps to generate an expertise matrix. That is, all the identified architectural elements and quality attributes within the identified design decisions in a project are used to create a matrix that indicates the architectural expertise of architects and developers in that project. As shown in Figure 4.3, this expertise matrix highlights the architectural expertise of individuals on specific architectural concepts. Furthermore, the same expertise matrix is also used to recommend suitable experts to address new design concerns.

The final step in the extraction process is creating cluster models of similar design decisions. This step uses an unsupervised ML technique to create a cluster model of similar design decisions in a project. That is, in this step, groups of similar design decisions are created and persisted in ADeX’s knowledge base. So that, a new open design concern can be matched against the cluster model to recommend end-users the most similar design decisions made in the past. This allows end-users to investigate how similar design concerns were addressed in the past, which components were impacted while addressing similar concerns, as well as estimate the time and effort involved in addressing similar concerns.

All the aforementioned steps are automatically handled by the underlying system without the involvement of end-users. End-users only have to select the project they would like to analyze through the UI. Once, for instance, all the issues are processed by various components in the framework, the visualizations discussed before are generated and presented to the users. In the next chapter, these steps are elaborated in detail with explanations about how to design and implement different components within the framework.

---

## System design and implementation

---

This chapter describes how each of the individual components is designed, implemented, and integrated to realize an instance of the AKM framework, which is referred to as ADeX. The UML 2.0 specification is used for the representation of the system's design diagrams. In the subsequent sections, first, the design is explained from the holistic system's perspective and then, each of the components within the system is elaborated.

As shown in Figure 5.1, the high-level architecture of ADeX follows the three-layered architectural style for web applications. The presentation layer comprises of independent web front-ends that provide UIs for end-users, administrators, and facilitators of the system. The end-users of ADeX mainly interact with the UI component named Amelie - Decision Explorer client. The other web clients for SyncPipes, document classifier, and document clustering provide UIs for administrators and facilitators to monitor and configure various parameters to enable the pre-processing of data from different data sources that maintain project related information. Next, the middle application layer comprises of multiple components including SyncPipes server, Akre-Server, Document Classifier, and Workbench4DC that handles the business logic of the system. The Akre-Server component is responsible for orchestrating the invocation of services offered by other components within the application layer. Finally, the persistence layer serves the data to the application layer. The persistence layer comprises of a model-based system called SocioCortex<sup>1</sup> that not only provides backend-as-a-service (BAAS) but also supports user management with its UI. The persistence layer also contains an instance of MongoDB<sup>2</sup> which is used to store various configuration parameters that are used by the components in the business logic layer.

The components in the presentation layer interact with the components in the application layer over HTTPS on predefined ports. The components in the application layer interact with the

---

<sup>1</sup><http://server.sociocortex.com/>

<sup>2</sup><https://www.mongodb.com/>

## 5. System design and implementation

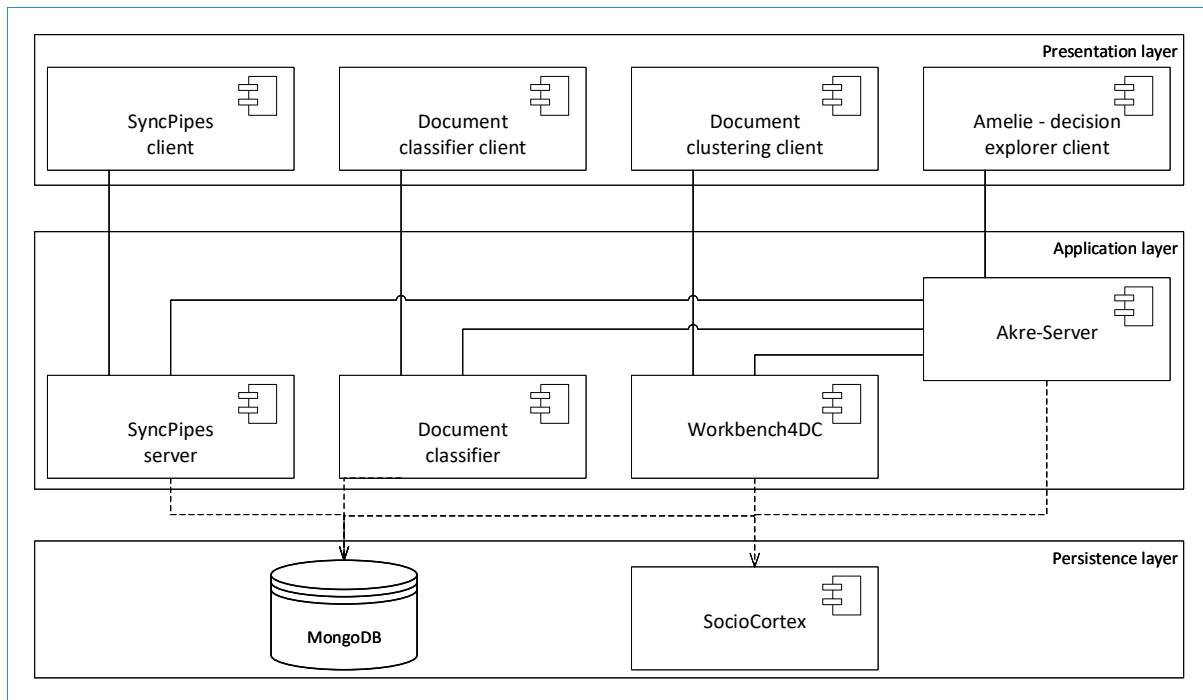


Figure 5.1: The high-level architecture of ADeX: the core components and their dependencies

persistence layer to read and write data. After performing the corresponding business logic, the response is sent back to the respective UI component. The design of ADeX also reflects the microservices architecture. Each of the components has its own UI in the presentation layer, has respective business logic in the application layer, and has corresponding collections in the MongoDB. Hence, every component can be deployed independently of each other and can communicate with each other through web services. Each of the components responsibilities along with their design is elaborated in the subsequent sections.

### 5.1 SocioCortex

As shown in Figure 5.1, SocioCortex acts as a backend of ADeX. SocioCortex not only maintains software projects' data but also captures the conceptual (domain) model of software engineering projects. The main benefit of using SocioCortex as a backend-as-a-service (BaaS) is that it provides domain experts and facilitators of ADeX the flexibility to configure and adapt the domain model at runtime. The runtime adaptation of the domain model helps to meet the needs (for example, naming conventions) of different software engineering projects. As alternatives, one can also use other systems such as Parse<sup>3</sup> or Apache Usergrid<sup>4</sup> to achieve the same capability.

SocioCortex is used as a collaborative information management system that enables stakeholders to manage their application's data as well as its conceptual model. This section describes the core concepts of SocioCortex and explains how the AK base is modeled in SocioCortex.

<sup>3</sup><https://parseplatform.org/>

<sup>4</sup><http://usergrid.apache.org/>

### 5.1.1 The hybrid-wiki meta-model

The meta-model of the SocioCortex platform is referred to as the hybrid-wiki meta-model [6,98]. An excerpt of this meta-model is shown in Figure 5.2. The *Entity Type* and *Attribute Definition* allow the definition of the domain model (also called the user model) specific to an application domain. The *Entity Type* is used to instantiate a domain-specific concept, and the *Attribute Definition* captures the properties of that concept. An *Attribute Definition* is further associated with a specific type and multiplicity constraints. A user can create a concrete domain model at runtime by instantiating the *Entity Type* and *Attribute Definition*. For instance, in the context of AKM, concepts such as Decision and Architectural Element are defined as *Entity Types*, and their attributes and relationships are captured using *Attribute Definitions*. Furthermore, users can create multiple *Entities* instantiating a specific *Entity Type*. An *Entity* can contain multiple *Attributes* relating to an *Attribute Definition*. The *Entity* and *Attribute* concepts allow users to create instances of the domain model that were captured as *Entity Types* and *Attribute Definitions*. All the above concepts are part of a so-called *Workspace* which acts as a container and restricts the scope. Once the domain model is captured in a *Workspace*, the model can be cloned, reconfigured, and reused for similar subsequent projects. The meta-model based platform supports the following use cases: (a) the domain model should be organization-, domain-, and project-specific and (b) the domain models should be reusable and configurable to the project's context (team size, development methodology, processes, etc.).

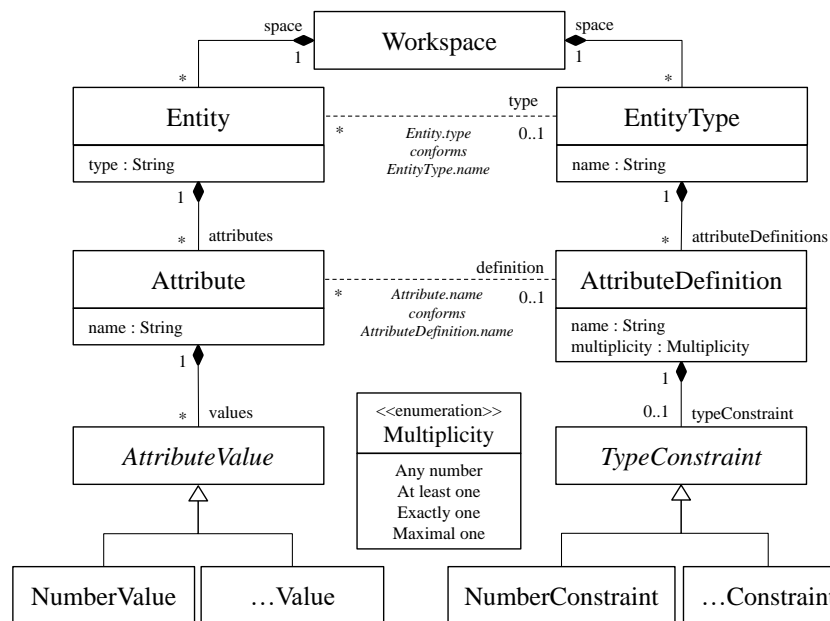


Figure 5.2: The hybrid-wiki meta-model of SocioCortex: image taken from [6]

### 5.1.2 Using SocioCortex for architectural knowledge management

Architectural knowledge can be classified into four broad categories, namely *context*, *design*, *general*, and *reasoning* knowledge [75]. The *context knowledge* captures project-specific information such as management information and architectural significant requirements. The *design knowledge* comprises of the architectural design of the system. The context and the design knowledge evolve as a software project progresses and hence they are referred to as **dynamic AK**. The general AK captures architectural methods, styles, patterns, and organization-specific corporate information (for example, organizational processes) that help architects while designing software systems. Since the general AK changes less frequently over the course of a project, we refer to general AK as **static AK**. Finally, reasoning knowledge contains information that guides software architects to apply static knowledge in the project context. It also maintains design decisions, rationale, and alternatives that were considered during the ADM process in specific project instances, which can be reused in similar projects.

The high-level design of how SocioCortex is used for capturing AK is shown in Figure 5.3. SocioCortex’s meta-model is used to create domain models (static and dynamic knowledge model) at runtime. The facilitators of ADeX can configure the domain model through the web UI of SocioCortex. The reasoning knowledge is managed by the rule engine component which captures the reasoning logic using a model-based expression language (MxL) that accesses the meta-model for computations. The domain models, model-based expressions, and model instances (data) can be managed through the REST APIs. Such a design ensures the “separation of concerns” design principle between client applications and the SocioCortex platform.

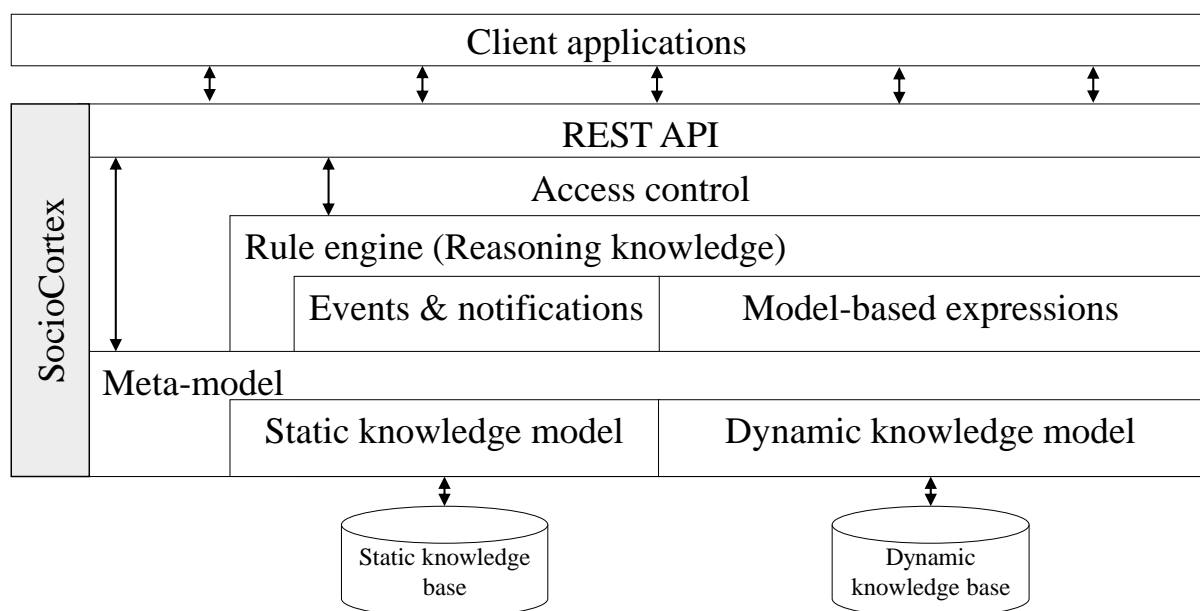


Figure 5.3: The design of an AKM system using SocioCortex as the backend

### 5.1.3 The dynamic architectural knowledge model

For capturing dynamic AK in software projects, the domain model comprises of concepts from project management, requirements engineering, AKM, implementation, and maintenance phases. Providing a consolidated view over the current state of the different SDLC phases is crucial for software architects during decision making. There exists a large body of knowledge that captures the concepts and relationships between them in different phases of the SDLC through general-purpose ontologies and addresses a multitude of problems including traceability, decision support, tool integration, and documentation. For instance, [7] propose an ontology to capture the structural elements of software architecture. Tang et al. present a software ontology for annotating knowledge in requirement and architecture documents and to support traceability and reasoning over requirements specification and architecture design [8,9,99].

Furthermore, efforts from industries to standardize the interfaces for tools used in the SDLC are becoming prominent, wherein multiple industry partners are coordinating to define standards such as the Open Services for Lifecycle Collaboration (OSLC) [10]. The OSLC standard not only has an underlying ontology but also provides guidelines for exposing data using the linked-data format with well-defined standard interfaces. Even though tool vendors have started to adopt such standards, the expressiveness of such standards is still limited (as it is still in the definition phase) and needs to be adapted for specific use cases. For constructing our domain model, we analyzed the existing ontologies [7–9], OSLC standard, and data models of specific tools such as Business Canvas Model, MS Project<sup>5</sup>, Enterprise Architect<sup>6</sup>, JIRA<sup>7</sup>, and Bugzilla<sup>8</sup> which are extensively used in practice.

The core concept within the domain model is *Project* with multiple attributes such as name, description, start date, and end date. As shown in Figure 5.4, a *Project* is associated with its corresponding *Business plan* which is further associated with concepts derived from the Business Model Ontology [100]. The updated version of this ontology (Business Model Canvas [101]) has nine core concepts including value proposition, partners, key activities, resources, and customers. The concept *Requirement* references the concept *Key activity* in the business model ontology. A project has multiple requirements which are classified into *Functional* and *Non-functional requirements*. As also modeled in [8] and [9], an architectural *Decision* depends on the requirements and results in a specific *Architecture*. A decision is made by considering multiple *Design alternatives*. An *Architectural rationale* justifies the decision made by an architect (cf. [4]). The architecture is further elaborated with concepts such as architectural *Element* composed of *Attributes* and *Methods*. These concepts are derived from the Enterprise Architect<sup>9</sup> modeling tool’s schema. Furthermore, since a software architect needs to have a consolidated view over the current project plan and the availability of resources, we have included the concepts from the project management domain in our domain model. The concepts and the relationship between the concepts such as *Person*, *Task*, and *Assignment* are derived from the work presented in [11] and the XML schema of the Microsoft Project management tool. Furthermore, in

---

<sup>5</sup><https://products.office.com/en/project/project-and-portfolio-management-software>

<sup>6</sup><https://sparxsystems.com/products/ea/>

<sup>7</sup><https://www.atlassian.com/software/jira>

<sup>8</sup><https://www.bugzilla.org/>

<sup>9</sup><https://sparxsystems.com/>

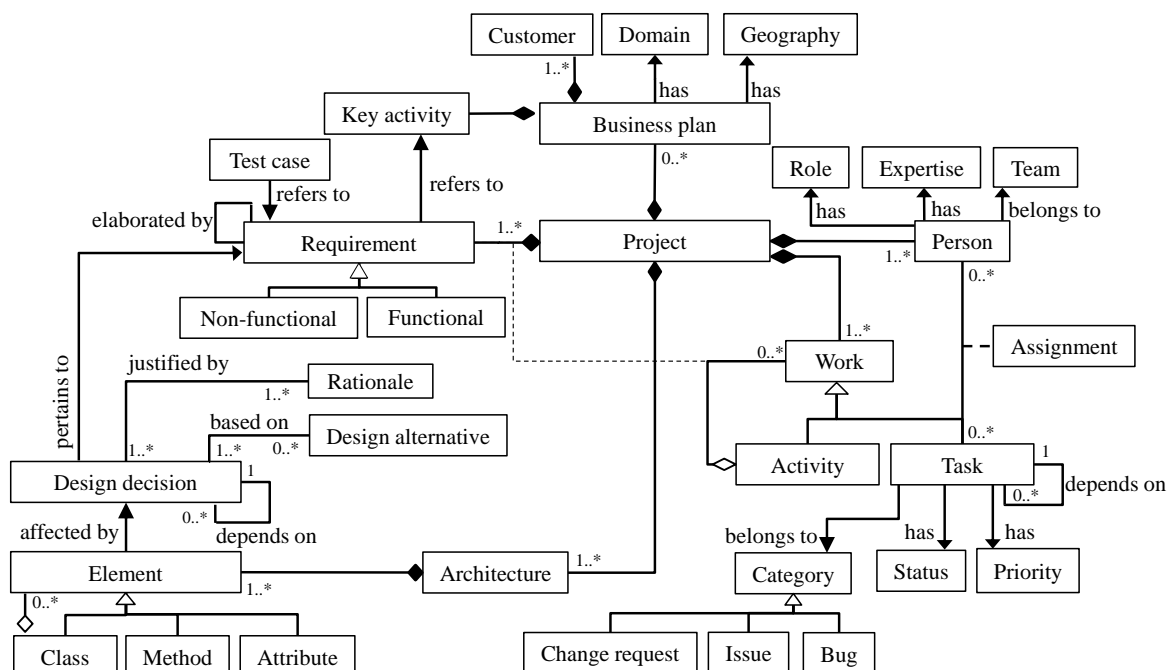


Figure 5.4: The dynamic AK model for capturing the design and the context knowledge (the concepts have been taken from [7–11])

SDLC, tools such as JIRA and Bugzilla are commonly used for managing tasks; these tools were also investigated. Concepts such as task *Categories* (*Issue*, *Bug*, and *Change request*) and their corresponding workflows derived from issue management systems were modeled in the dynamic architectural knowledge model. To keep the view of the domain model simple and to allow better readability, not all the concepts and the relationships are presented in Figure 5.4. It should be noted that the domain model is an instance of the meta-model and hence it can be configured through the SocioCortex’s UIs and can be adapted as per the project needs.

In this dissertation, we emphasize on the concepts related to design decisions shown in the lower-left corner of Figure 5.4. Given that most of the concepts and relationships shown in Figure 5.4 are derived from the existing literature, our main contribution to the model is establishing the relationships between the concepts related to design decisions and the software engineering artifacts. To the best of this author’s knowledge, this is the first attempt to model design decisions in such a holistic context.

#### 5.1.4 The static architectural knowledge model

The static AK captures knowledge about architectural styles, reference architectures, design patterns, architectural methods, and architectural standards. Furthermore, it also captures organization-specific corporate knowledge such as templates for architectural methods, experts who can help instantiate architectural methods in a specific context, and the time and cost involved in applying architectural methods and standards. The model of static architectural



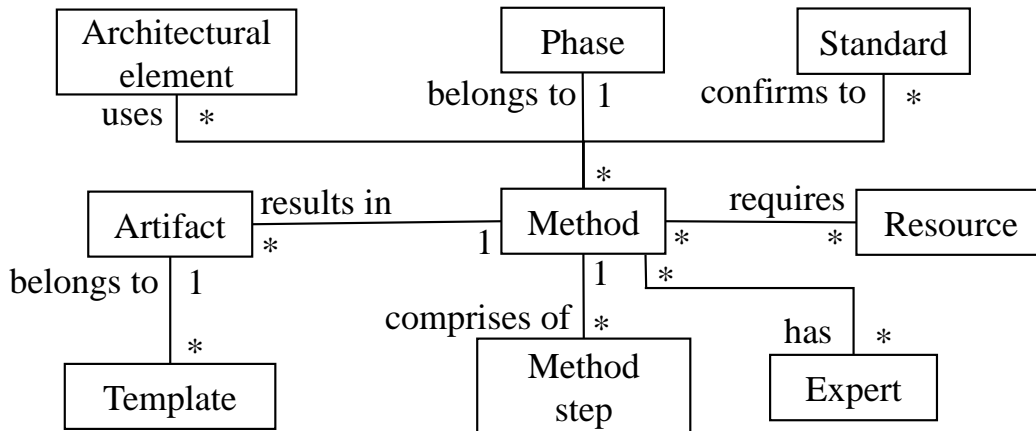


Figure 5.5: The static AK model for capturing general AK

knowledge is shown in Figure 5.5. This model is again an instance of the hybrid-wiki meta-model presented in subsection 5.1.1, which allows domain experts to extend the model with new concepts at runtime in SocioCortex.

The static knowledge model focuses on the *Methods* applicable in the architectural lifecycle of a project. Examples of architectural methods include design methods (attribute-driven design, 4+1 views, and Siemens four-views [102]) or analysis methods (scenario-based analysis, architecture analysis method, and architecture trade-off). The architectural method belongs to a *Phase* in the architecture life cycle and can be composed of multiple *Method steps*. A *Method* confirms to a specific architectural *Standard* and requires human, budget, and time *Resources*. Within an organization, architects (*Experts*) with expertise in specific methods can help other project partners in applying those methods. Furthermore, specific *Methods* could also use *Architectural elements* such as architectural styles and patterns in their method steps. Each *Method* generates corresponding *Artifacts* which can be instantiated using organization-specific *Templates*.

The knowledge base comprising of both the static and dynamic AK which is modeled in SocioCortex covers the following use cases:

- Create, persist, modify, and delete AK (that represents both the static and dynamic AK) elements in the knowledge base.
- Search AK elements using keywords and categories.
- Subscribe to AK elements and get notifications when they are updated.
- Manage different versions of the AK.

The need for these use cases is also highlighted in [76]. The focus of the knowledge base component is to manage (gather, structure, store, search, and version) AK and to make them available for all the other value-added services that guide software architects based on the current state of the project.

### The rule engine component

Using information about the current state of a project (which is available from the dynamic AK model) and the application-generic knowledge (static AK model), the rule engine evaluates the rules in the rules repository and presents relevant recommendations to the end-users. These rules are event condition action (ECA) rules implemented using the model-based expression language (MxL) [103]. The MxL is a domain-specific language that allows querying the domain model in SocioCortex. A rule represented using MxL is an expression and executing a rule implies evaluating the corresponding expression. Since MxL is defined over the hybrid-wiki meta-model, it can access all the EntityTypes - which represent concepts in the domain model. A simple expression such as “*find(Project).where(Status='ongoing')*”, returns all the instances of Projects that are currently ongoing. For implementing such rules, one can also use alternative business rules management systems such as Drools [104]. However, these rule engines are typically model-based and not meta-model based systems. In other words, a change in the domain model would also require updating the rules, as well as, the source code of the system. Whereas, MxL is implemented over the meta-model of the SocioCortex platform. A change in the domain model is automatically reflected in the MxL rules. For instance, if the concept “Task” is changed to “Issue” in the domain model then, the respective MxL rules are updated accordingly.

An event in an ECA rule triggers the execution of the corresponding rules. These events are categorized into three broad classes as described below.

- *Domain events* are triggered due to data-level changes within the artifacts of a project. Typically, these are the frequently occurring events since they reflect the current state of a project. The domain events could represent, for instance, uploading an architecture review document, changing a task’s status in a project, or adding a decision to use a specific reference architecture. Furthermore, since data is captured as instances of Entity and Attribute types in the hybrid-wiki meta-model (cf. subsection 5.1.1), operations (create, update, or delete) on these types would trigger a domain event.
- *Model-change events* are triggered when the domain model is updated to accommodate the changing project context. For instance, when a domain expert deletes the relationship between Decision and Architecture and adds a new relationship between Decision and Architecture Element, the corresponding rules need to be updated and re-evaluated. Since concepts belonging to the domain model are represented as instances of Entity Type and Attribute Definition in the meta-model, any operation (update and delete) on them will trigger a model-change event.
- *User-triggered events* are used when an architect actively executes (triggers manually) the rules to identify the actions that need to be performed.

If the condition in an ECA rule is satisfied, the corresponding action is performed. Conditions are expressed using MxL expressions and can be either simple or nested expressions. For instance, a simple expression could include checking the status of a project and a nested expression could represent multiple such queries joined by ‘and’ and ‘or’ operators. If all the conditions in an ECA rule are satisfied, the corresponding actions are recommended to the end-users. These actions are represented as MxL expressions as they allow invocation of operations on the data.

Amelie » Static Functions » STATIC::getDomainExperts

### Custom MxL Function **STATIC** :: getDomainExperts

Name	getDomainExperts
Description	Recommend domain experts
Parameters	projectName:String
Return Type	Sequence<String>

Method Stub

```
let projectVar = find(Project).single(name=projectName) in
let task = find(Task).single(name="prepare architecture documentation" and project =
projectVar) in
let conditions = if projectVar.status = "ongoing" and projectVar.Phase = "Design" and
task.Status = "closed" then true else false in
let actions = find(Expert).where(Expertise = "Architecture review and analysis" and Experience
> 5).select(name) in
if conditions then actions else []
```

Figure 5.6: An exemplary MxL expression for finding domain experts based on an ECA rule

To illustrate the application of a simple rule, consider the following use case: the system should recommend domain experts with a minimum of five years of experience in architecture review and analysis. Furthermore, such a recommendation should be context-aware, that is, it is relevant for software architects only when the following conditions are satisfied:

- c1) the status of the project is “ongoing”,
- c2) the project is in the “design” phase, and
- c3) the task “prepare architecture documentation” is “complete”.

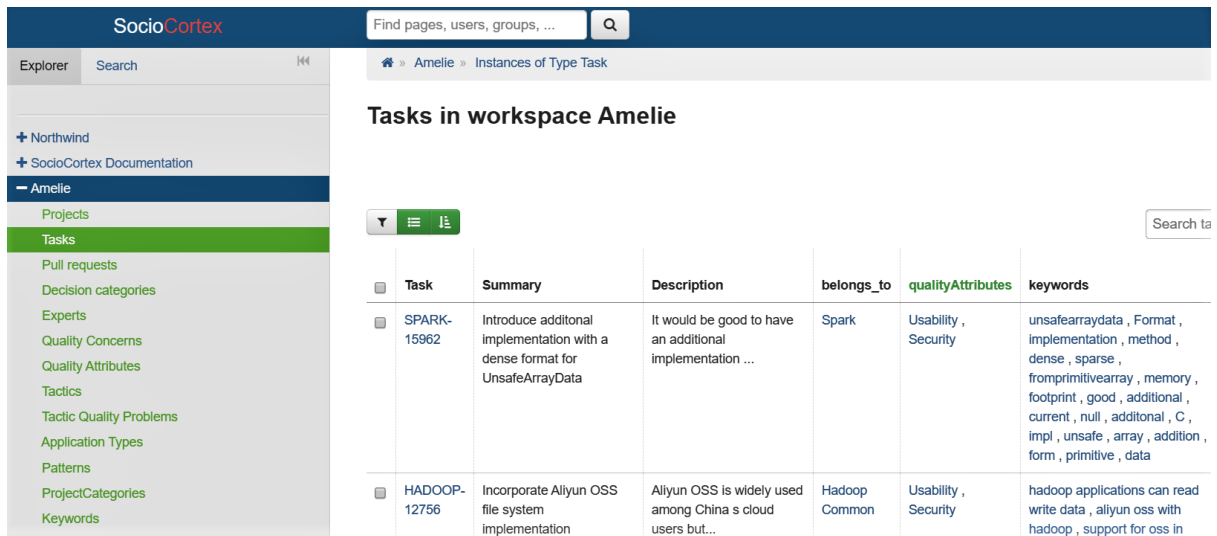
The corresponding ECA rule can be captured as an MxL expression along with the meta-information such as name, description, and parameters as shown in Figure 5.6. The parameters are a list of input variables that can be used within an expression.

The above rule is evaluated every time there is a domain event corresponding to a Project or a Task. This rule can also be triggered by a user-triggered event and by executing the *getDomainExperts*(‘Amelie’) expression. This example shows how the system uses the dynamic knowledge about the current state of the project along with the static knowledge (with expertise in the generic knowledge base) to recommend context-sensitive information to software architects. It also indicates that the expressions rely on the concepts from the domain model introduced in Figure 5.4. The rules in the repository not only include such recommendations but also include actions such as generating reports or downloading templates based on the decisions taken by the architects during the project. The rule engine component specifically addresses the use case to offer automated support to software architects and developers during the ADM process.

### 5.1.5 User interface for capturing architectural knowledge in SocioCortex

SocioCortex is a collaborative web-based application that allows (a) domain experts to model an application’s domain as well as (b) end-users to capture and maintain data adhering to

## 5. System design and implementation



The screenshot shows the SocioCortex web application. On the left is a navigation sidebar with a search bar and a list of workspaces: Northwind, SocioCortex Documentation, and Amelie (selected). Under Amelie, there are categories like Projects, Tasks (selected), Pull requests, Decision categories, Experts, Quality Concerns, Quality Attributes, Tactics, Tactic Quality Problems, Application Types, Patterns, ProjectCategories, and Keywords. The main content area is titled 'Tasks in workspace Amelie' and contains a table with two rows of task data.

Task	Summary	Description	belongs_to	qualityAttributes	keywords
SPARK-15962	Introduce additional implementation with a dense format for UnsafeArrayData	It would be good to have an additional implementation ...	Spark	Usability , Security	unsafearraydata , Format , implementation , method , dense , sparse , fromprimitivearray , memory , footprint , good , additional , current , null , additional , C , impl , unsafe , array , addition , form , primitive , data
HADOOP-12756	Incorporate Aliyun OSS file system implementation	Aliyun OSS is widely used among China s cloud users but...	Hadoop Common	Usability , Security	hadoop applications can read write data , aliyun oss with hadoop , support for oss in

Figure 5.7: The user interface of SocioCortex for collaboratively modeling the application domain and for capturing the instances of the domain concepts

that domain model, collaboratively. Here, “collaboratively” refers to the idea that both domain experts and end-users learn from each others’ activities and perform their tasks in an environment where both the model and its data co-evolve [6].

Figure 5.7 shows the UI of SocioCortex<sup>10</sup>. The list of workspaces in SocioCortex is made available in the navigation bar on the left side of the screen. On selecting a specific workspace, the workspace is expanded within the list of Entity Types within that workspace. In Figure 5.7, the workspace named Amelie is selected which contains Entity Types such as Project, Task, Quality Attributes, Decision Category, etc. These Entity Types and their relationships have been created by domain experts for maintaining AK in software projects. On selecting an Entity Type, all the Entities (data/instances) belonging to that Entity Type are presented in a data table. In Figure 5.7, all the instances of Tasks are shown in the center of the screen. The data table presents all the attributes and their corresponding values. For example, corresponding to the Entity Type - Task, the data table shows the task’s name, its description, the project to which it belongs to, if it is a design decision, and if so, the category of a design decision. End-users can create and update these attributes through the UI.

As discussed in the next section, all the tasks of a project are automatically extracted from issue management systems such as Jira and Github Issues using the SyncPipes component and are persisted in SocioCortex. Those issues are then processed to automatically extract architectural knowledge. For example, as shown in Figure 5.7, each task is automatically labeled to indicate if it is a design decision or not a design decision; they are also annotated with architectural elements (see concepts and keywords in Figure 5.7) and quality attributes.

<sup>10</sup><http://server.sociocortex.com>

## 5.2 SyncPipes

The SyncPipes component addresses the general use case of AK integration for AKM tools. Specifically, this component synchronizes data that is spread across software engineering lifecycle tools to one centralized Knowledge Base (KB). SyncPipes acts like a “bot” with sensors and actuators for our platform. The sensors monitor the current state of projects’ artifacts, and actuators keep the information within the KB synchronized. Figure 5.8 illustrates the conceptual model of the SyncPipes component. The **Source** model represents various source tools that can be handled by SyncPipes (e.g., JIRA, MS Project, Excel, and Enterprise Architect); whereas, the **Target** model corresponds to the KB of ADeX. Both the source and target models are specializations of **ToolModel**. The ToolModel provides interfaces to connect with the tools using user credentials. It also implements methods such as `getTypes` and `getEntities` to work with a unified data-model representation. The data model of the source and target tools conform to a meta-model represented using the JSON-Schema [105] format. The SyncPipes component uses a **Mapper** to capture the mapping of the concepts from the source to the target tool’s data model. For instance, a facilitator of ADeX can map a concept “New Feature” in JIRA to “Requirement” in ADeX’s domain model. The mapper allows facilitators to consider only those concepts and attributes that are relevant and necessary to establish traceability across the artifacts. Once a facilitator specifies the mapping, the **Handler** concept uses the mapping information to extract data from the source tool, to transform the data, and to persist it in the target tool. Subsequently, the Handler also starts the **SyncJob** that triggers the transformation process based on specific events or at regular time intervals. The concepts mentioned above support selective model selection, alignment, transformation, and synchronization of data residing in different tools into ADeX.

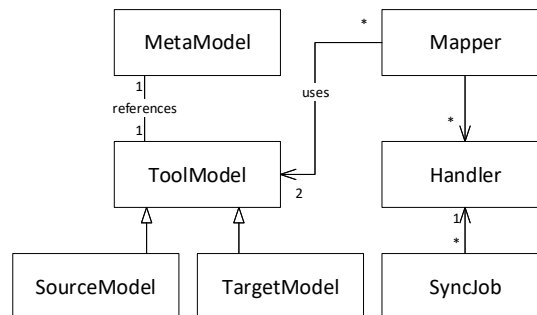


Figure 5.8: The conceptual model of SyncPipes for AK integration and synchronization

The core aspect of the SyncPipes component is the extensibility of the data integration services to include new source and target systems. This is achieved through the so-called services which comprise of several interfaces that developers must adhere to while implementing new services. Figure 5.9, provides an overview of the interfaces necessary for implementing new extractor and loader services. Each extractor service is responsible for fetching data from individual data sources. An extractor service could either be active or passive. An active extractor service fetches data from data sources that expose their data through REST APIs. Once a reference API is

configured, such active services do not require human intervention for updating data during the synchronization process. Whereas, a passive extractor service corresponds to extracting data from files such as Microsoft Excel or Comma Separated Value (CSV) files. On the other hand, a loader service is responsible for importing the data extracted from the extractor service into the corresponding target system. The most essential interfaces necessary for implementing a new extractor or loader service are described below.

### 5.2.1 The configuration of the extractor and the loader services in SyncPipes

For a service that requires custom configurations such as credentials for accessing a database or REST APIs, the service must include a custom configuration that will be an instance of ServiceConfiguration. The service configuration describes the structure of the configuration data using the JSON-Schema format. For example, the configuration of a loader service to import issues from JIRA includes - JIRA's root API, user name, password, and a project key. This configuration also implements a getConfigSchema method that returns the data schema of the configuration. The configuration of a service is persisted in the database using the store method. The load method is used to fetch the persisted configuration data from the database into the service's configuration.

#### The extractor and the loader services in SyncPipes

SyncPipes is a platform for developers to write extractor and loader services based on the interfaces defined in SyncPipes. The end-users of SyncPipes can then choose their desired source and target systems, configure the extractor and loader service, define the mapping between the source and target data structures, and finally start the execution process. To achieve this, developers must implement all the interfaces shown in Figure 5.9.

The extractor and the loader services have the same base interface named Service. The getName method returns the name of the service which is used as a primary key and is exposed through a REST API to the SyncPipes client. The getConfiguration method returns the associated ServiceConfiguration instance. On the other hand, the setConfiguration method sets a specific configuration that will be used during the extraction or the loading process. Each service

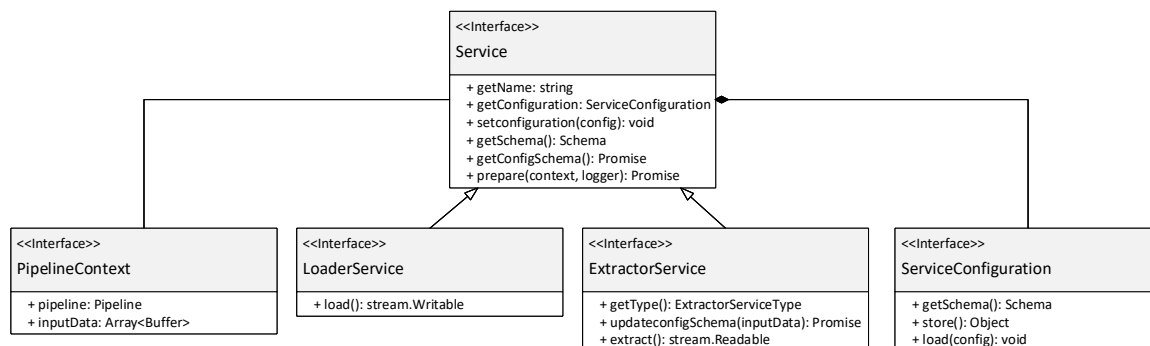


Figure 5.9: Interfaces for implementing the extractor and loader services in SyncPipes

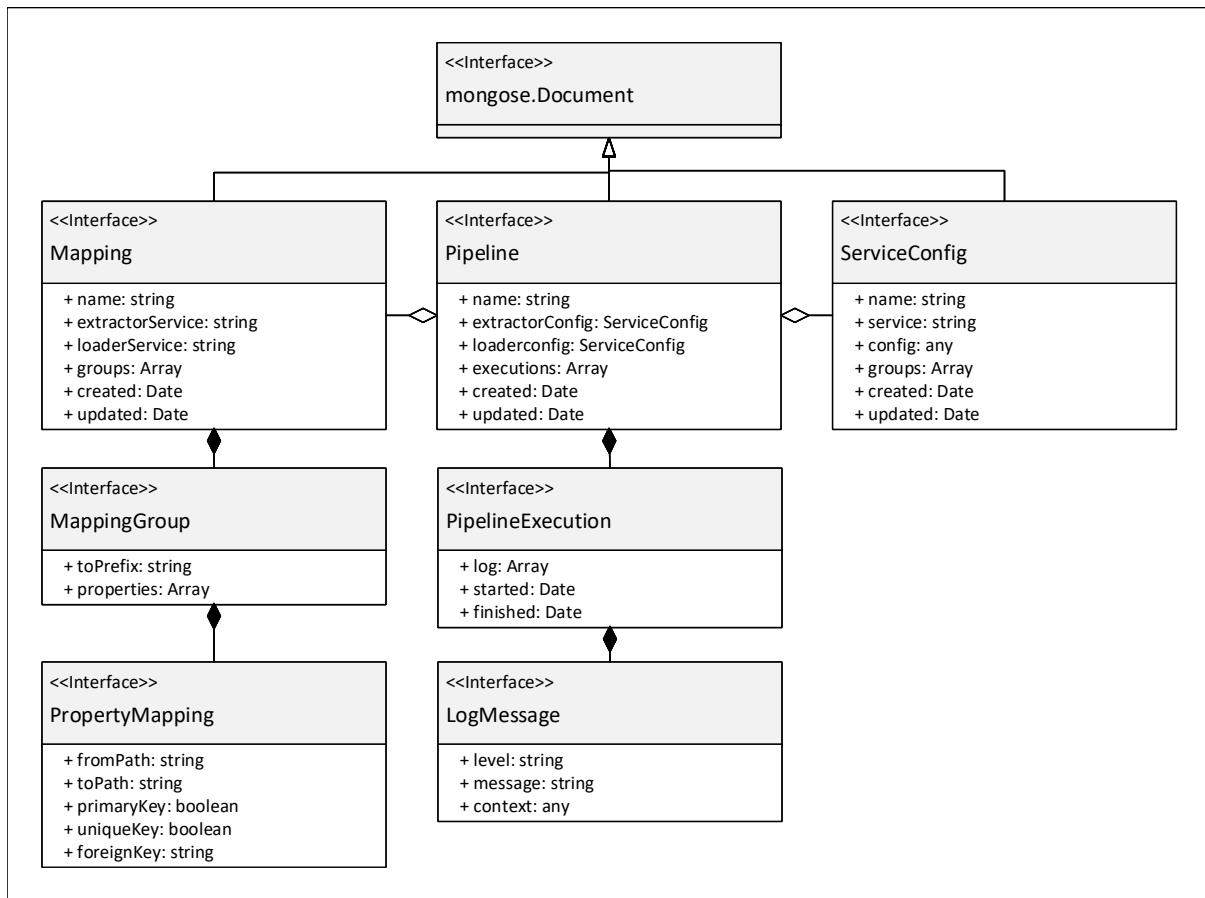


Figure 5.10: Concepts for persisting ETL pipelines and data mappings in SyncPipes

describes the data that it either generates as output (for extractors) or expects as input (for loaders) using the JSON-Schema. The `getSchema` method returns the data schema that a service is generating or expecting. If the data schema is unknown at compile time, then it needs to be updated using the `updateSchema` method during the execution time in the `prepare` method. The `prepare` method is invoked right before starting the extraction or the loading process. An instance of the `PipelineContext` is passed along with an instance of the `Logger` interface to the `prepare` method. The `PipelineContext` has two properties, namely, `pipeline` and `inputData`. A pipeline that needs to be executed is an instance of the concept – `Pipeline` (cf. Figure 5.10). The `inputData` is an array of `Buffers`<sup>11</sup>, which is only available for passive extractor services. The second argument passed to the `prepare` method is an instance of `Logger` which enables services to write log messages that are associated with the current pipeline execution. The `prepare` method performs those tasks which are required before starting the actual extraction or loading processes. For example, this could include connecting to a database or setting up an HTTP client. A service that extracts data from a source system has to implement the `ExtractorService` interface, which extends the `Service` interface with two additional methods, namely, `getType` and `extract`. The `getType` method returns a value of the `ExtractorServiceType` which can

<sup>11</sup><https://nodejs.org/api/buffer.html>

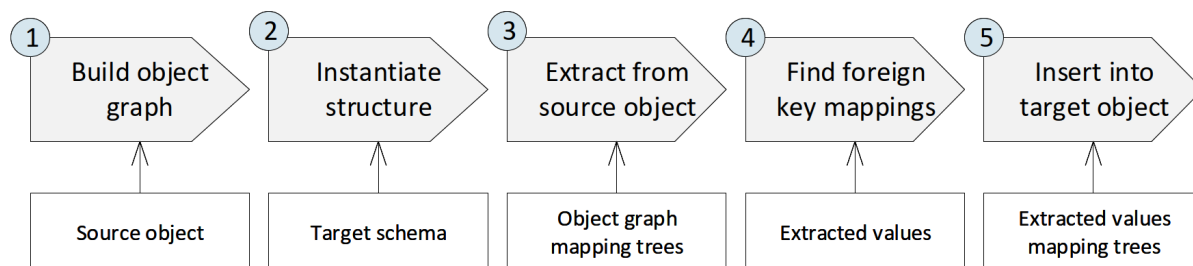


Figure 5.11: The steps in the data transformation workflow in SyncPipes

either be active or passive. If the type is `Passive`, the application expects to be provided with additional data, for example, an Excel file. The data is made available to the service through the `PipelineContext.inputData` property. Second, an `ExtractorService` implements the `extract` method that returns a `stream.Readable`<sup>12</sup>. The extractor service then uses this stream to push the data extracted from the source system.

Subsequently, the loader services allow importing the extracted data into the target system. To implement a loader service, the developer has to implement the `LoaderService` interface that also extends the `Service` interface. The interface requires the implementation of the `load` method, which returns an instance of the `stream.Writable`<sup>13</sup>. Through that stream, a loader is provided with the extracted data from the extraction service which will be loaded into the target system.

### 5.2.2 Data transformation using user-defined data mappings

For importing data from the source system into the target system, users can map the data schema of the source system with the data schema of the target system. Based on the mapping, SyncPipes performs the actual data transformation. Three fundamental elements are required to perform the data transformation process:

1. Both the extractor and the loader services must describe the data schema they either produce or expect using the JSON-Schema.
2. A mapping to transform the data between the source and the target system.
3. Data which will be transformed; this data must adhere to the corresponding JSON-Schema.

As shown in Figure 5.10, every pipeline has an associated mapping. It is a meta-data container encapsulating the name of the mapping, names of the corresponding extractor and loader services, and the dates when the mapping was created and updated. Each mapping is broken down into groups, which are further broken down into properties. This allows end-users to map the nested tree structure of the JSON data-schema from the source system to the target system. The `PropertyMapping` interface has two properties `fromPath` and `toPath`. The `fromPath` property defines which values should be extracted from the source object and the extracted values are then inserted into the target object using the `toPath` property.

<sup>12</sup>[https://nodejs.org/api/stream.html#stream\\_class\\_stream\\_readable](https://nodejs.org/api/stream.html#stream_class_stream_readable)

<sup>13</sup>[https://nodejs.org/api/stream.html#stream\\_class\\_stream\\_writable](https://nodejs.org/api/stream.html#stream_class_stream_writable)



The GraphTransformer handles the data transformation. It is instantiated with the schemata of the extractor and the loader service as well as with an instance of the respective Mapping. The graph transformer first converts the mapping into a tree structure and then transforms the instances of the source schema using the provided mapping. As shown in Figure 5.11, the transform method consists of five steps; the arrows are the different steps in the transformation process, and the boxes describe the required input data for each step.

In Step 1, the graph transformer generates a tree structure of the input source's data schema using the composite design pattern. In Step 2, it creates an initial tree structure of the target system's data schema. Next, in Step 3, using the mapping's tree structure, the graph transformer extracts the corresponding source data. Next as part of the insertion step, first the foreign keys are handled by inserting the primary keys into the target object in Step 4. Finally, in Step 5, the remaining extracted values are inserted into the target object.

The end-to-end workflow of the SyncPipes data extraction and loading process is shown in Figure 5.12. Transforming data using an extractor, and a loader services requires executing a Pipeline, which is selected by the end-user of the system. If the requested pipeline exists, the associated services are loaded and the application creates a new pipeline execution object and persists it in the database. Thereafter, a new pipeline context is created and is populated with the information related to that pipeline, which is serialized and sent to a message broker (RabbitMQ). The message broker notifies the GraphTransformer to start a new pipeline execution. On receiving the pipeline context as a message, the GraphTransformer loads the services

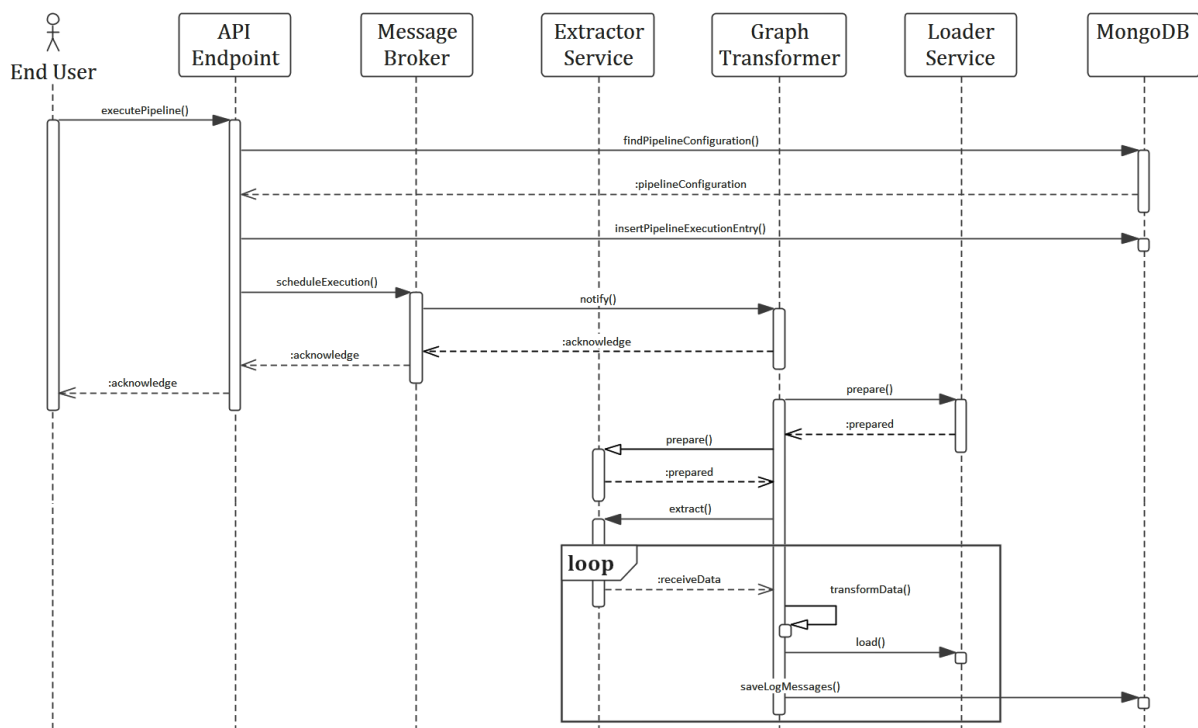


Figure 5.12: Sequence diagram showing the data extraction and loading process in SyncPipes

associated with the pipeline. Then, the prepare method is called both on the extractor and the loader services. The GraphTransformer waits until the prepare methods are complete. Upon completion of the prepare method, the GraphTransformer invokes the extractor service. The GraphTransformer receives the data from the extractor service and pushes it to the mapper. Using the respecting mapping configuration, the mapper transforms the data provided by the extractor service and pushes it to the loaded service. Finally, the loader service invokes the load method that loads the data extracted from the source system into the target system. Figure 5.12 captures the aforementioned steps using a sequence diagram.

### 5.2.3 User interfaces for managing data transformation

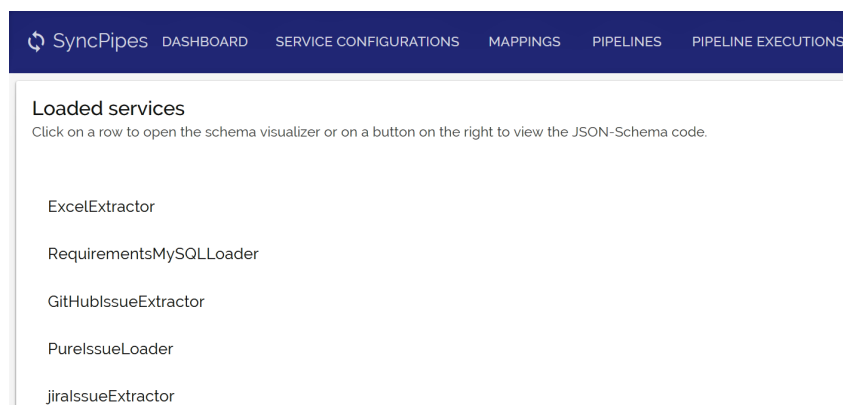


Figure 5.13: The dashboard view in SyncPipes shows the available extractor and loader services as well as the status of the ETL jobs

The SyncPipes client component provides facilitators of ADeX, the UIs to configure the pipelines for importing and synchronizing data from different systems into a target system. The SyncPipes client is a front-end web application implemented using TypeScript<sup>14</sup> which transcompiles to JavaScript<sup>15</sup>. The SyncPipes client interacts with the SyncPipes server using REST APIs to invoke the exposed services and to present the results in the UIs (cf. Figure 5.1).

Using the dashboard view of the SyncPipes client, facilitators of ADeX can quickly view all the available extractor and loader services. For example, as shown in Figure 5.13, SyncPipes provides extractor services such as ExcelExtractor, JiraIssueExtractor, and JiraProjectExtractor services. It also provides loader services such as MySQLLoader, MongoLoader, and SocioCortexLoader services. Furthermore, on this dashboard, the facilitator can even glance over the previously executed integration pipelines and see the status of those executions.

As discussed in the previous sections, each of the extractor and loader services has associated configurations. These configurations can be updated by the facilitator using the interface shown in Figure 5.14. Depending on the service that a facilitator wants to use, she can select the respective service and edit its configuration. Figure 5.14 shows the configuration parameters for the JiraIssueExtractor service. These parameters include the *description* of the configuration,

<sup>14</sup><http://www.typescriptlang.org/>

<sup>15</sup><https://developer.mozilla.org/en-US/docs/Web/JavaScript>

Edit service configuration ×

Select a service  
jiraIssueExtractor ▾

If a option is disabled, the service is not configurable.

Description  
Extract issues from jira

url  
issues.apache.org/jira

username  
sebis@test.com

password  
\*\*\*\*\*

project  
CSV

Figure 5.14: The user interface for configuring an extractor or a loader service in SyncPipes

*URL* of the Jira repository that exposes REST APIS, *user name* and *password* to access those APIs, and finally, the *name of the project* in the Jira repository to extract the issues. Once these parameters are configured, they can then be reused for executing multiple pipelines.

Furthermore, it should also be noted that, when the end-users of ADeX select a project to analyze using the Amelie - Decision Explorer client (discussed later in Section 5.6), the aforementioned configuration is automatically created using the configuration API provided by the SyncPipes server and the extraction process is automatically executed.

Once the configurations for the extractor and loader services are created, next, the facilitator can generate the mapping between the data model of the extractor service and the loader service. The mapping between the data models is created in a declarative manner using the UI shown in Figure 5.15. On selecting the extractor and the loader service, along with their configurations, their corresponding data models are presented to the facilitator. Figure 5.15 shows the data model of the extractor (JiraIssueExtractor) service on the left side. On the right side, the data model of the loader (MongoDBLoader) service is presented. The tree structure of the data model can be explored to view the attributes within each node (concept). Facilitators select and map the nodes as well as their attributes (on both sides). For better understandability and readability, the mapping is also shown on the lower part of the screen. For example, the attribute *summary* of an *issue* type in the extractor service's data model is mapped to the attribute named *title* of *task* type in the loader service's data model.

Once such a mapping is created between an extractor and a loader service, it is further used as a mapping configuration while importing the data into the target system. It should be noted that such a mapping needs to be created only once between the extractor and loader service. For

## 5. System design and implementation

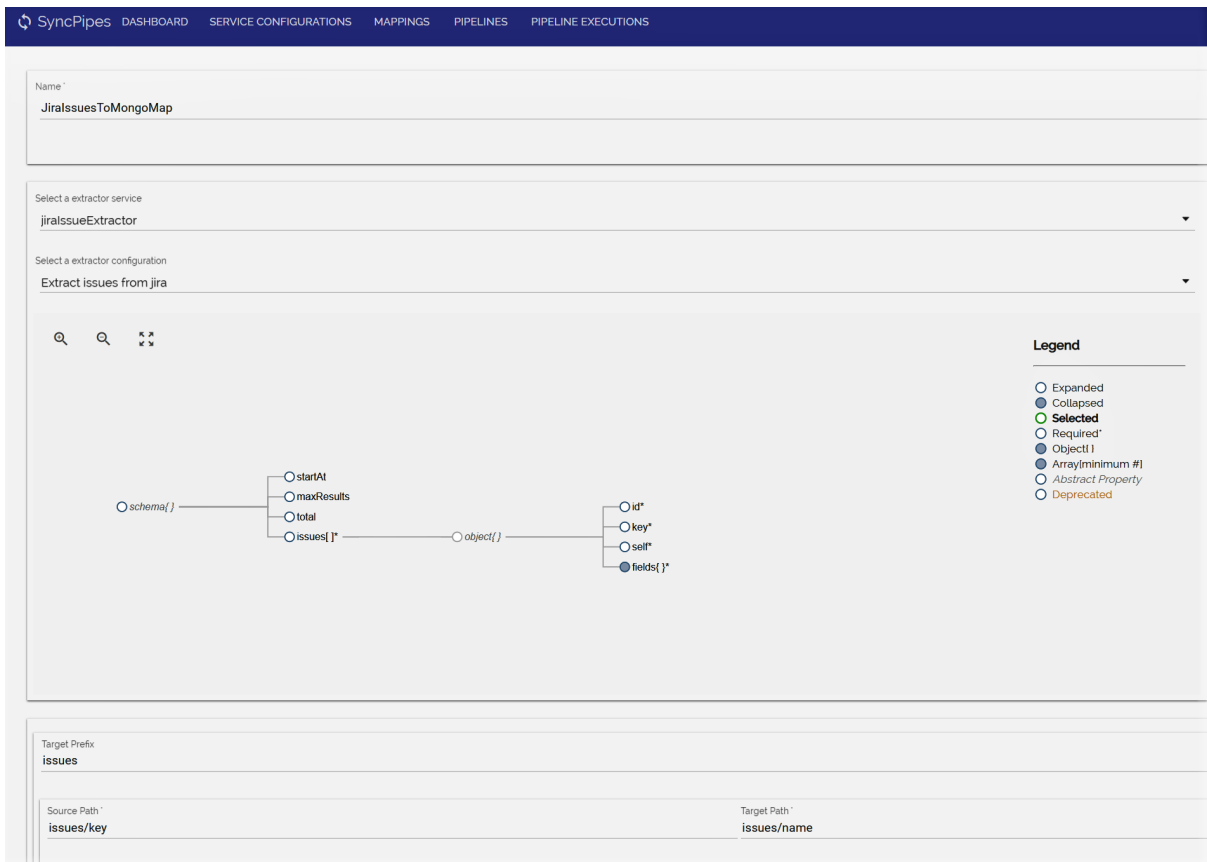


Figure 5.15: The user interface for configuring a pipeline’s data mapping in SyncPipes

instance, while importing issues from Jira into SocioCortex, the same mapping configuration is used for all the projects even though, the execution pipeline is different for each of them.

Finally, when both the extractor’s and loader’s service configurations and the mapping between the data models of those services are available, an execution pipeline can be created as shown in Figure 5.16. This execution can either be executed by clicking on the play icon on the UI or can be invoked by calling the corresponding REST API exposed by the SyncPipes server. The later is used by the ADeX system to trigger the import process automatically.

SyncPipes was developed as part of Frido Koch’s Bachelor thesis project [12].

Pipelines							CREATE NEW PIPELINE
Name ↑	Mapping	Extractor Config (Service)	Loader Config (Service)	Passive	Created	Updated	
GitHub Inversify to MySQL	<a href="#">GitHub to MySQL</a>	GitHub Inversify (GitHubIssueExtractor)	Local MySQL (PureIssueLoader)	No	06/18/2016 6:35:32 PM	06/18/2016 6:35:32 PM	

Figure 5.16: The user interface for editing and executing a pipeline in SyncPipes [source: [12]]

### 5.3 Document classifier

The Document Classifier component addresses the use case of *identifying and classifying design decisions from issues* extracted using the SyncPipes component. Once all the issues in a project are imported, a two-phase supervised ML-based approach is used to identify design decisions within those issues. As shown in Figure 5.17, in the first phase, an issue is either classified as a “design decision” or “not a design decision” using the decision detector component. The decision detector component uses a classification model that has been trained using more than 1,500 labeled issues from two open source projects, namely, Apache Spark and Apache Hadoop. If the decision detector component labels an issue as a design decision, then that design decision is further processed by the decision classifier component in the second phase. The decision classifier component uses a pre-trained classification model to further classify the decision into one of the three decision categories, namely, structural, behavioral, and non-existence or ban decisions.

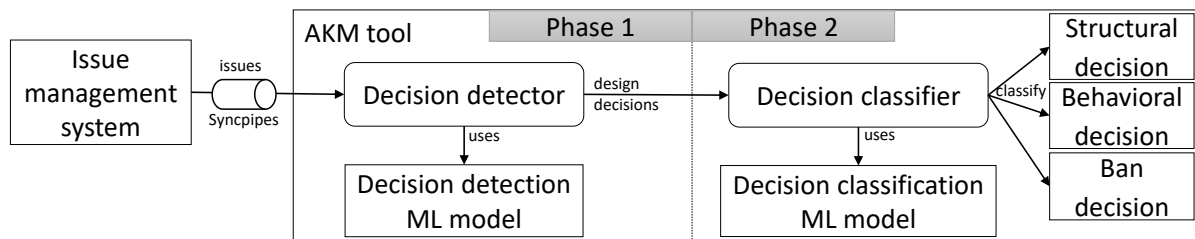


Figure 5.17: The high-level overview of the design decision detection and classification process

The Document Classifier component is implemented using the Java Play web application framework<sup>16</sup> and has its own three-layered architecture. The presentation layer provides the necessary UIs for administrators of ADeX to create classification pipelines, run those pipelines, and to analyze the execution results. These functionalities can also be invoked by other components in ADeX (specifically, the Akre component) through the exposed web-service APIs. The application layer provides the logic to train and run the ML classification models. The data (issues) necessary for training the model is imported from the backend of ADeX (SocioCortex). Finally, the persistence layer uses a document-store (MongoDB) to store the information related to the classification pipelines and all the classification models are persisted on the filesystem as shown in Figure 5.18. Each of these layers are subsequently elaborated.

#### Persistence layer of the document classifier component

In the context of classifying documents, a classification pipeline includes a series of steps necessary for generating a ML classification model which is trained using labeled training dataset. All the relevant information corresponding to a pipeline is persisted in a document-oriented database (MongoDB). As shown in Figure 5.19, a clustering pipeline is uniquely identified by its name. Each pipeline has a set of labels that can be assigned to a new document that needs to be classified. Furthermore, for each label, a path or a location to the set of training documents must be provided. These training documents can either be on the filesystem or can be tables or collections in a database. Apart from the labels, those attributes of a document that must be used for training the classification model must also be provided. These attributes are referred

<sup>16</sup><https://www.playframework.com/>

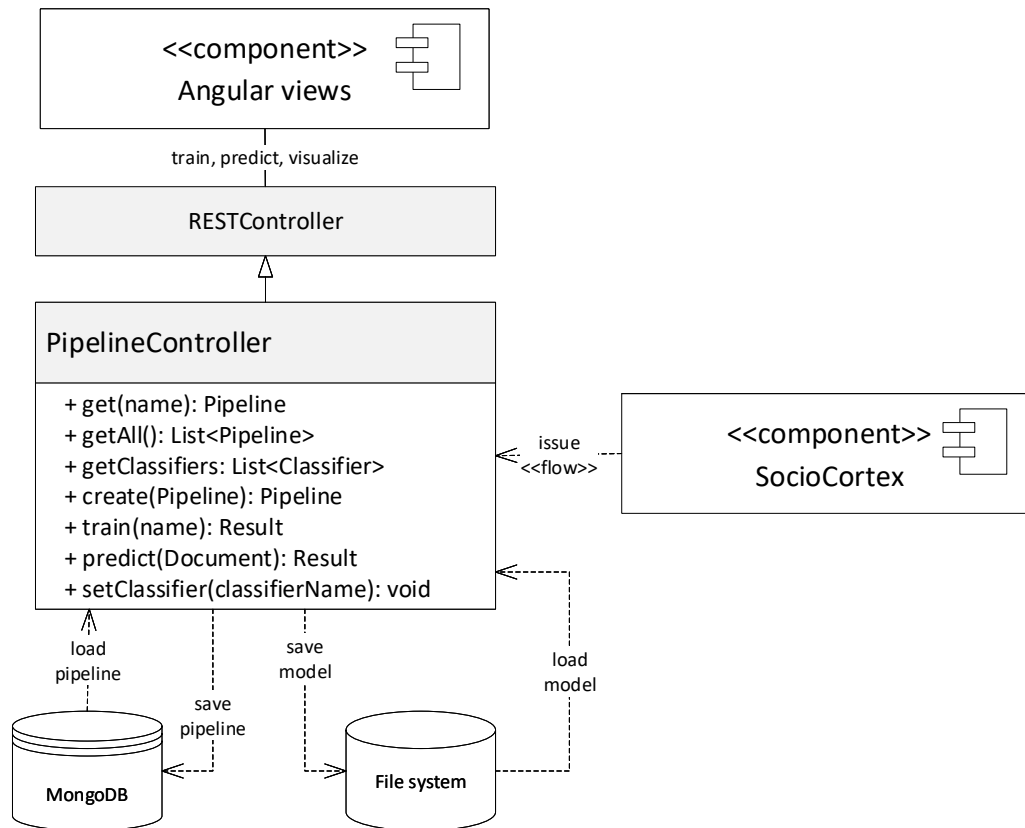


Figure 5.18: The high-level system design of the document classification component

to as mining attributes. For instance, the summary and description attributes of issues in an IMS are used for training the Decision Detection model. The trained model is persisted in the filesystem using the `modelPath` attribute in the classification pipeline.

For training a classification model, users can specify the classifier that needs to be used. The current implementation of Document Classifier supports two classifiers, namely, Naive Bayes and Support Vector Machine (SVM) classifiers. The selection of a classifier is maintained in the classifier attribute in the respective clustering pipeline.

### Application layer of the document classifier component

The logic for **training** the classification models and for predicting the label for a new document is contained in the application layer. To train a classification model, first, a pipeline is created using the create functionality in the Pipeline controller. While creating a new pipeline, users must provide a classifier, labels for the classification, and the location of the labeled documents. Once a pipeline is created, the “train” functionality can be called upon, which invokes the training pipeline’s run method to load the training data, preprocess them, build a classification model using the respective classifier, save the model in the filesystem, and finally return the classification results.

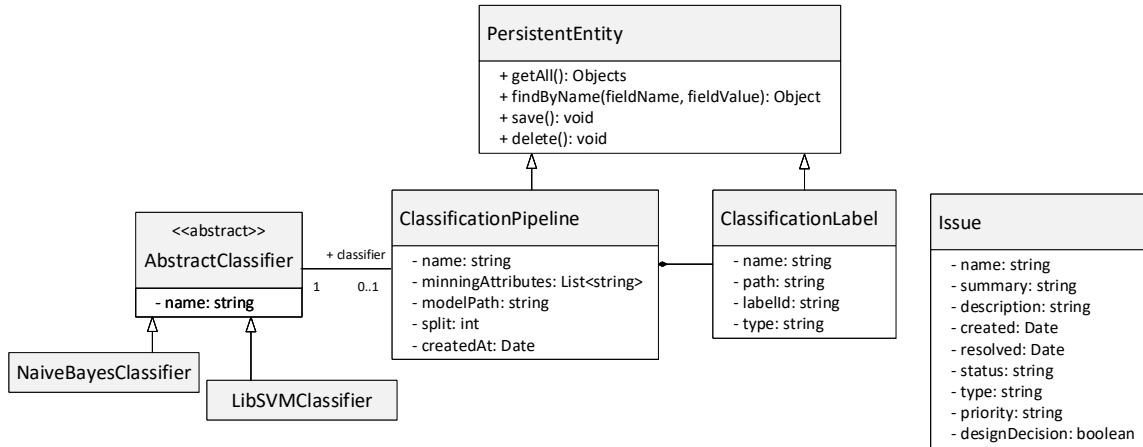


Figure 5.19: A class diagram showing the concepts and their relationships in the document classification component

The execution of a training pipeline includes a series of standard ML steps. For detecting design decisions in issues, each issue’s textual description (summary and description) is considered as one document. As discussed in the previous sections, a labeled training dataset with 1,571 issues (760 design decisions and 751 non-design decisions) is used to create the decision detection model. Once the training dataset is loaded and transformed into instances consumable by the Weka library, a sequence of standard NLP steps are applied for preprocessing the documents. As shown in Figure 5.20, these steps include, tokenizing the documents, transforming those tokens to lower cases, removing stops words, stemming the tokens to their root words, and finally, generating token pairs using n-grams. Once, the preprocessing is complete, for each document, a tf-idf vector representation is created. These documents are then split into training and testing datasets (using split strategies - cf. Evaluation chapter). The training dataset is then used to create the classification model using 10-fold validations. Once the model is created, it is persisted on the filesystem, and the testing dataset is used to check the accuracy (F-score) of the classification model which is sent back to the client for further analysis.

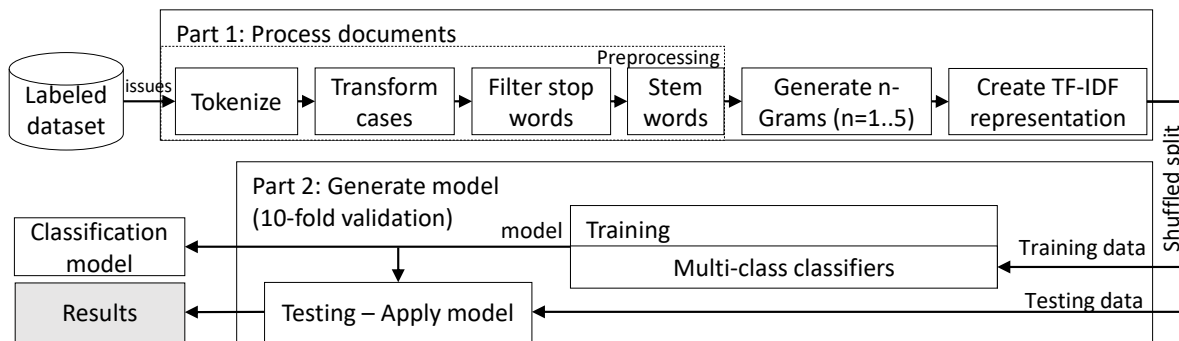


Figure 5.20: A machine learning pipeline for design decision detection and classification

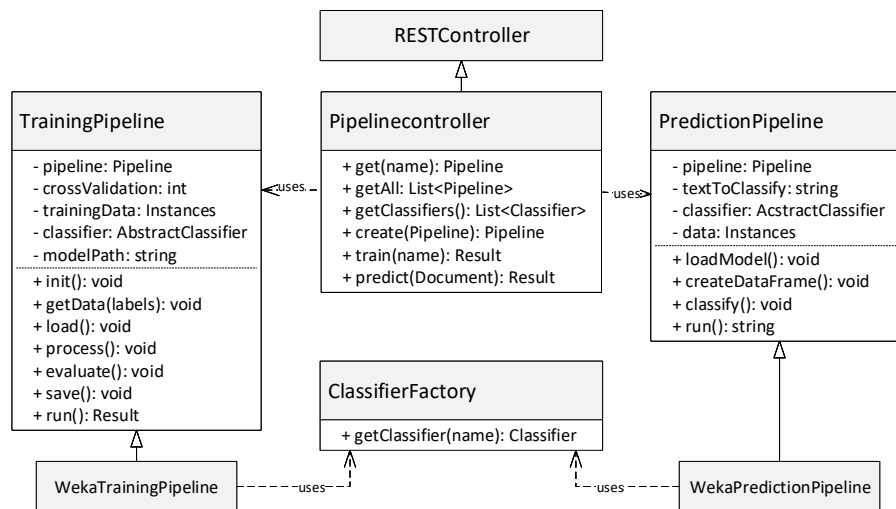


Figure 5.21: The application controllers and their dependencies for managing business logic in the document classification component

The same steps are applied for generating a classification model for classifying design decisions into one of the three categories, namely, structural, behavioral, and non-existence or ban decisions. The only difference being that the labeled training dataset is different. Among the 760 labeled design decisions, 360 design decisions are labeled as structural decisions, 300 as behavioral, and 250 as ban design decisions. These three labels with the corresponding datasets are used for creating the trained decision classification model.

For **predicting** a label of a new document in the application phase, the predict function in the Pipeline controller is invoked. The Pipeline controller uses the Weka prediction pipeline, which is a specialization of prediction pipeline (cf. Figure 5.21) to process the new document. The prediction pipeline first loads the persisted trained classification model from the filesystem, creates a data frame of the new document that can be processed by the Weka library, preprocesses the document, finds the appropriate label, and sends back the label to the client.

### Presentation layer of the document classifier component

The UIs in the presentation layer allow administrators and facilitators of ADeX to manually create new classification pipelines, configure the classifiers, train the classification models, and analyze the results of classification.

## 5.4 Workbench4DC: Document clustering component

The Document Clustering component addresses the use case of *identifying similar design decisions made in the past*. Once all the issues in a project are labeled by the Document Classifier component as either “design decision” or “not a design decision”, the Document Clustering component aggregates all the design decisions belonging to that project, builds a cluster model,



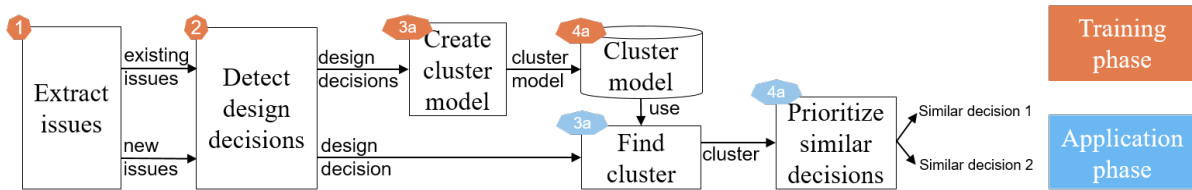


Figure 5.22: The high-level overview of the document clustering process

and persists it on the file system. These cluster models are created using the K-means algorithm. Hence, each cluster contains a set of similar design decisions around its centroid. The aforementioned steps correspond to the *training phase* in the ML process and are depicted in Figure 5.22. During the *application phase*, for a new design decision within that project, the Document Clustering component loads the already created cluster model, identifies the cluster to which the new design decision belongs to, and then ranks the most similar documents within that clustering using a document similarity measure. The resulting similar decisions are then returned to the Amelie - Decision Explorer client and presented to the end users. Note that, in the subsequent discussions, a document refers to a design decision and clustering of documents corresponds to the clustering of design decisions.

### System design of the document clustering component

The Document Clustering component is self-contained and has its own three-layered architecture with presentation, application, and persistence layers. As shown in Figure 5.23, the persistence layer handles the storing and retrieval of the ML pipelines, its configurations, the cluster models, and the results of pipeline executions. The application layer handles training the cluster models and predicting the clusters for new design decisions. Furthermore, it also exposes the necessary

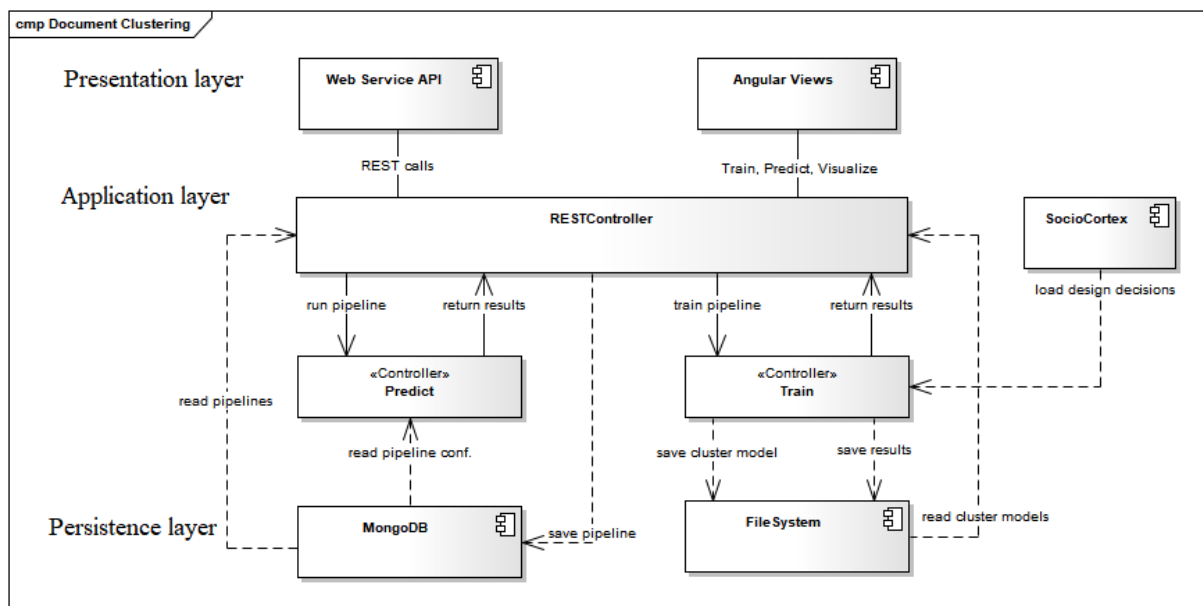


Figure 5.23: The high-level system design of the document clustering component

application programming interfaces (APIs) which are accessed by the presentation layer as well as by other components in the ADeX system. The UI in the presentation layer allows administrators and facilitators of ADeX to configure, run, and analyze the results of executing a clustering algorithm to find similar design decisions.

**Persistence layer of the document clustering component**

Document clustering uses MongoDB and the filesystem for storing information. The information regarding the pipeline configurations and available libraries are stored in MongoDB. On the other hand, the trained models and their corresponding results are stored on the filesystem.

In the context of clustering documents, a cluster pipeline includes the selection of:

- a ML library (for example, Apache Spark or Weka)
- a clustering algorithm (for example, K-means or bisecting k-means)
- configuration options (for example, the desired number for k and number of iterations)

When a user or an external system (through the exposed REST APIs) creates new cluster pipelines for training the documents in a project, the corresponding information is stored in MongoDB using the collections shown in Figure 5.24. Once the clusters are created, the corresponding models are stored in the filesystem, and the name of the project is used as an identifier. During the application phase, that is, given a new design decision, the system loads the saved model using the project name, retrieves the respective cluster pipeline from MongoDB, and finally executes the “predict” method to identify similar design decisions.

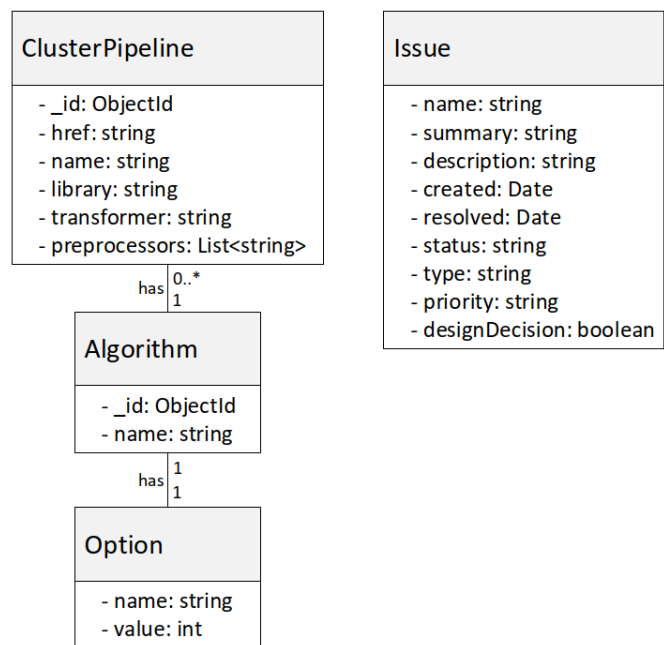


Figure 5.24: A class diagram showing the concepts and their relationships in the document clustering component

## Application layer of the document clustering component

The controllers in the application layer handle the logic of creating, training, and applying the cluster models to new documents. Hence, this layer comprises of two central controllers, namely, Train controller and Predict controller as shown in Figure 5.23. To create and train the cluster models for all the design decisions in a project: the Training controller first loads all the design decisions from SocioCortex, uses the configurations set by the user to create the cluster models, and persists the results before sending the results to the presentation layer.

The Predict controller is responsible for all operations related to predicting the cluster for a new design decision and identifying most similar design decisions within that cluster. For a new design decision in a project, the Predict controller first loads the saved cluster model from the filesystem, applies the model on the new design decision, and finds the cluster label for this new design decision. Once the cluster label is identified (which contain many design decisions), a ranking algorithm using the Cosine similarity is applied to each design decision within that cluster to return an ordered list of similar design decisions.

The functionalities to train a cluster model and to predict similar decisions are exposed as

The screenshot shows the 'Workbench4DC' interface for creating a new pipeline. The page title is 'Workbench4DC' and the navigation menu includes 'Create Pipeline', 'Cluster Documents', and 'Cluster Visualization'. The main heading is 'Create new pipeline'. Below this, the pipeline name is 'DocClustering'. The 'Pipeline name:' field is empty. The 'Select Library' section has 'Apache Spark' selected. The 'Select Clustering Algorithm' section has 'KMeans' selected, with 'Bisecting-KMeans' as an alternative. The 'Select feature extractor' section has 'Word2Vec' selected, with 'Hashing-TF' as an alternative. The 'Set Algorithm Options' section has '20' for 'K-value' and '10' for 'iterations'. The 'Upload Data' section has two buttons: 'LINK SOCIOCORTX WORKSPACE' and 'LINK TO MONGODB'. Below this, the 'HADOOP' section has a 'Project Key' field and a 'BROWSE' button. The 'UPLOAD' button is also present. The 'Data Format' section has 'csv' selected. At the bottom, there is a 'SAVE & RUN' button.

Figure 5.25: The user interface for creating and configuring a document clustering pipeline

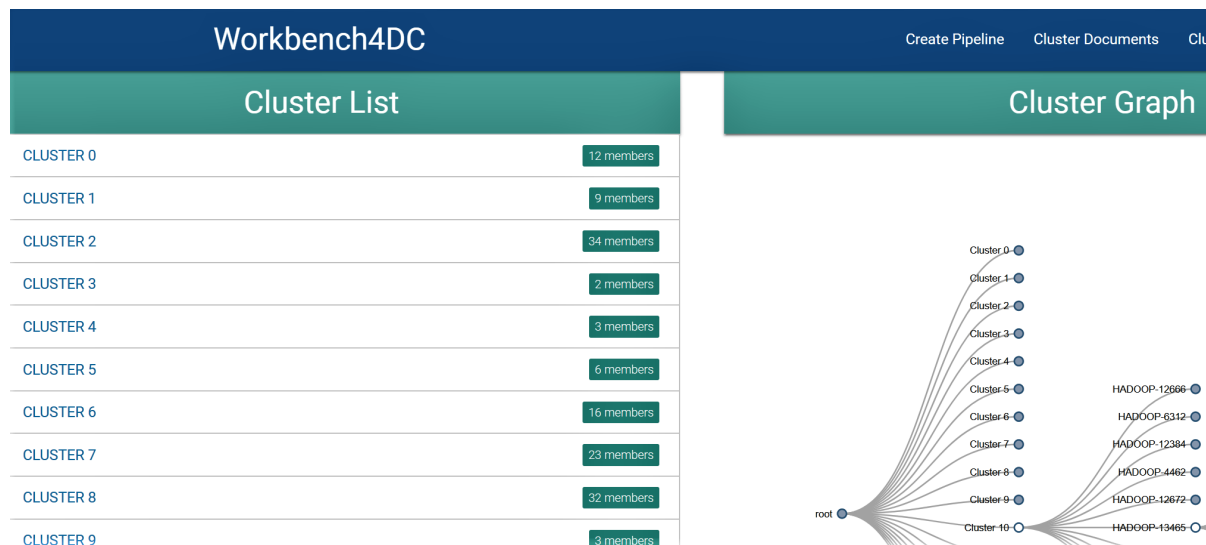


Figure 5.26: The user interface showing the result of a executing a document clustering pipeline

REST interfaces by the REST controller. These REST endpoints are invoked both from the UI of Document Clustering component as well as from the Akre-Server component that orchestrates the preprocessing of design decisions within a project.

### Presentation layer of the document clustering component

The UIs in the presentation layer allow administrations and facilitators of ADeX to manually create new cluster pipelines, configure the parameters necessary of creating cluster models, train the cluster models, and analyze the results of the generated clusters.

Figure 5.25 shows the UI for creating a new document clustering pipeline. An admin starts by providing a unique name for the pipeline followed by selecting the ML library (e.g., Apache Spark) for creating the cluster models. Then she can choose the clustering algorithm (KMeans<sup>17</sup> or Bisecting KMeans<sup>18</sup>) and the method for feature extraction (Hashing-TF<sup>19</sup> or Word2Vec<sup>20</sup>). Furthermore, the admin can also configure the value of K (which is the number of clusters) as well as the maximum number of iterations to run. Next, the admin can choose from either of the two data sources from where to retrieve the design decisions using a unique project key (either from SocioCortex or from MongoDB). In Figure 5.25, HADOOP is provided as the project key to link and extract the design decisions from the MongoDB. Alternatively, the admin can also choose to upload a CSV file for building the cluster model. Once all the configuration parameters are set, the admin can click on the “Save & Run” button to execute the pipeline.

Once the document clustering pipeline is executed, and the cluster model is generated, the admin is redirected to the UI shown in Figure 5.26. This view allows her to analyze the generated cluster model. On the left-hand side, the admin can see the list of clusters (20 based on the

<sup>17</sup><https://spark.apache.org/docs/latest/mllib-clustering.html#k-means>

<sup>18</sup><https://spark.apache.org/docs/latest/mllib-clustering.html#bisecting-k-means>

<sup>19</sup><https://spark.apache.org/docs/latest/mllib-feature-extraction.html#tf-idf>

<sup>20</sup><https://spark.apache.org/docs/latest/mllib-feature-extraction.html#word2vec>

**Workbench4DC** Create Pipeline Cluster Documents Cluster Visualization

## Predict Cluster Label

Pipeline: HADOOP

Support external calls in the [RPC](#) call queue. Leveraging [HADOOP-13465](#) will allow non-[rpc](#) calls to be added to the call queue. This is intended to support routing [webhdfs](#) calls through the call queue to provide a unified and protocol-independent [QoS](#).

**GET CLUSTER LABEL**

Predicted label: 10

DOC Id	DOC Key	Summary	Description	Cosine Similarity	Jaccard Similarity
198	HADOOP-13537	support external calls in the rpc call queue	leveraging hadoop will allow non rpc calls to be added to the call queue this is intended to support routing webhdfs calls through the call queue to provide a unified and protocol independent qos	100.00	90.00
141	HADOOP-13465	design server call to be extensible for unified call queue	the rpc layer supports qos but other protocols ex webhdfs are completely unconstrained generalizing server call to be extensible with simple changes to the handlers will enable unifying the call queue for multiple protocols	53.91	17.65

Figure 5.27: The user interface for predicting the the cluster model of a given design decision

configuration in Figure 5.25) and the number of documents in each cluster. On the right-hand side, the admin can select a specific cluster and view each of the individual document (only the ID) within each cluster. On selecting a document, she can further see the features (terms) that resulted in including that specific document in the respective cluster.

If the cluster models have a disproportionate number of documents within the clusters, the admin can navigate back to the create pipeline view, reconfigure the algorithm parameters, and rerun the clustering pipeline. Furthermore, the admin can test the generated cluster model by providing a new document as shown in Figure 5.27. Once, the admin provides the document and clicks on the “Get cluster label” button, the Predict controller uses the corresponding cluster model and predicts the cluster label for that document (cluster label is 10 in Figure 5.27). Moreover, the admin also sees the related similar documents within that cluster which are ranked according to the Cosine similarity score.

The document clustering component was designed and developed as part of Prateek Bagrecha’s Master thesis project [106].

## 5.5 Akre-Server: Architectural recommendations component

The Akre-Server component addresses multiple use cases, namely, a) identifying and annotating architectural elements within the textual description, b) recommending alternative solutions for an architectural element, c) detecting and annotating quality attributes within the textual description, and d) recommending experts to address new design concerns. Apart from addressing the aforementioned use cases, as also presented in Figure 5.1, this component also

serves as a middleware that synchronously invokes preprocessing services in different components (SyncPipes, Document Classifier, and Document Clustering), aggregates the results, and returns the response to the client application, which is, the Amelie - Decision Explorer client. Before discussing the invocation of preprocessing services, first, all the use cases are described below. Also, note that, unlike other components, this component does not have a dedicated UI. It only provides the business logic and interacts with the persistence layer.

### 5.5.1 Architectural solutions recommender

This sub-component uses the DBpedia ontology for identifying architectural elements within text and then identifies alternatives corresponding to those architectural elements. The high-level design of this sub-component is shown in Figure 5.28. The text that needs to be annotated (description of a design decision) can be provided as input from the Amelie - Decision Explorer client or through the invocation of the REST service.

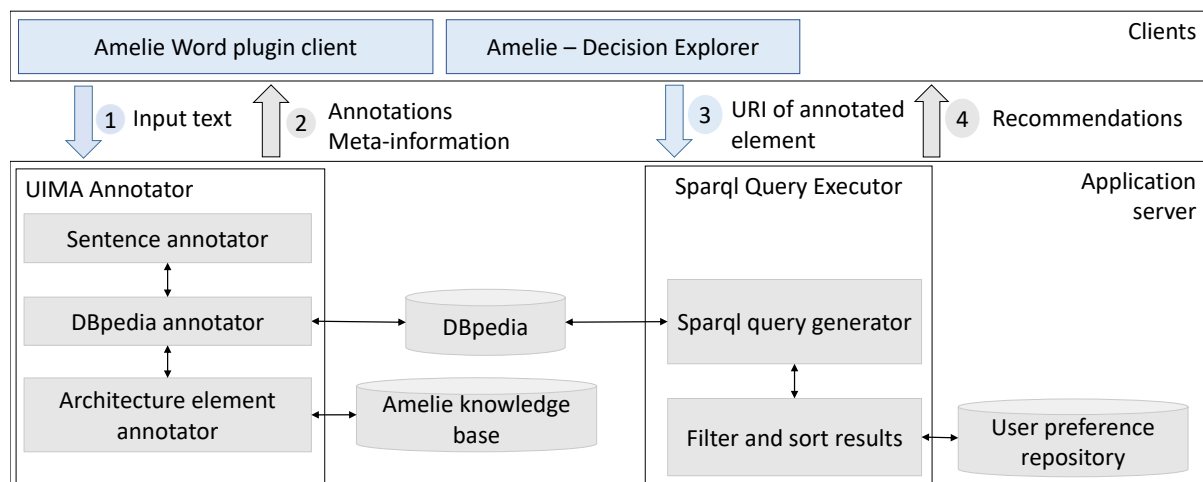


Figure 5.28: The high-level system design of the recommendation component

On receiving the textual information, this sub-component triggers a set of annotators that identifies and annotates architectural elements within the text and sends back those annotations to the client. Next, corresponding to an annotated architectural element, the lookup sub-component retrieves meta-information about that architectural element from the DBpedia ontology and sends it to the client. Subsequently, the SPARQL query executor identifies alternative architectural choices and returns the recommendations to the client (Step 3 in Figure 5.28). These steps are further elaborated in the subsequent subsections.

### 5.5.1.1 Phase 1: Automatic annotation of architectural elements

For annotating text with architectural elements, the annotators are implemented with the Unstructured Information Management Architecture (UIMA) framework<sup>21</sup>. This framework provides the necessary infrastructure to configure and run pipelines of annotator components for analyzing unstructured information. In the scenario under consideration, three annotators (namely, sentence annotator, DBpedia annotator, and architectural element annotator) are used.

1. **Sentence annotator:** The input text received from the client is broken down into sentences using the sentence annotator. These sentences are passed to the DBpedia annotator.
2. **DBpedia annotator:** For annotating text with concepts in the DBpedia ontology, a plugin named DBpedia Spotlight [89] is reused. DBpedia Spotlight is an open-source project that provides REST-based web services to annotate text with DBpedia entities. DBpedia Spotlight performs two main tasks: *phrase spotting* and *disambiguation*. In the first step, the phrase-spotting algorithm identifies phrases that should be linked to DBpedia entities in the given input text. These phrases are identified using a string-matching algorithm (Aho-Corasick<sup>22</sup>). In the second step, the phrases are further scored using TF-IDF weights and compared using Cosine similarity (cf. [89]). All the phrases with similarity scores above a configurable threshold value are returned to the client. The evaluation results presented in Section 6.1.1 are based on a threshold value of 0.7. The result from this annotator is further processed by the Architecture element annotator to generate a JavaScript Object Notation (JSON) array of annotations as shown in Listing 5.1.

Listing 5.1 shows the annotation of an architectural element – “relational database” which starts at character position 271 and ends at the 290<sup>th</sup> position within a given textual description. The start and end positions indicate the UIs to color code the architectural elements in the text. The similarity score is returned by the disambiguation step which indicates the relevance of the annotated resource. The support parameter (value: 625) indicates the number of incoming links to this particular DBpedia resource. The term that was annotated is also included in the JSON response as well the link to the DBpedia resource. The type parameter is discussed in the subsequent sub-sections.

```

1  {{
2    "begin": "271",
3    "end": "290",
4    "similarityScore": "0.9982600192890149",
5    "support": "625",
6    "URI": "http://dbpedia.org/resource/Relational_database",
7    "token": "relational database",
8    "type": "Genre"
9  }}

```

Listing 5.1: An exemplary JSON response with one annotation item

<sup>21</sup><https://uima.apache.org>

<sup>22</sup><http://alias-i.com/lingpipe>

- 3. Architecture element annotator:** To improve the precision of the annotated text results, an architecture element annotator is implemented which restricts annotations received from the DBpedia annotator to those that correspond to architectural elements. A bag of words consisting of tokens captured in the knowledge base is used for filtering out irrelevant annotations. Admins can create or update these concepts and tailor the list to specific projects. If an admin marks annotations as irrelevant, then such annotations are filtered out by this annotator. Alternatively, if the admin adds a custom annotation, then the corresponding annotation is added to the result. In the current prototype, these concepts are identified in text using a string-matching algorithm.

### 5.5.1.2 Phase 2: Retrieve meta-information for architectural elements

If an end-user is interested in a specific annotated architectural element (which is indicated by a mouse-click in the client), a new request along with the URI of the annotated element is sent to the Akre-Server. The look-up component in the Akre-Server uses the URI to retrieve properties of the architectural element from the DBpedia ontology. The list of properties, for instance, include `dbo:abstract`, `dct:subject`, `owl:sameAs`, `rdfs:comment`, `dbo:wikiPageID`, and `dbo:wikiPageExternalLink`. These properties within the DBpedia ontology (`dbo`) are derived from the upper ontologies including Dublin core (`dct`), Web Ontology Language (`owl`), Resource Description Framework Schema (`rdfs`), Simple Knowledge Organization System (`skos`), etc. The properties of the requested element are compiled into a JSON object and sent back to the client. The meta-information related to the selected architectural element is displayed on the right side of the UI under the meta-information tab within the ADeX client.

### 5.5.1.3 Phase 3: Recommendation support

To provide recommendations related to the architectural elements including architectural styles, patterns, and software solutions, concepts and relationships between concepts within the DBpedia ontology were first manually analyzed. Understanding these concepts and their relationships is necessary for realizing two specific types of recommendations (see below as Rec 1 and Rec 2). The subset of concepts within the DBpedia ontology, that is necessary for formulating the SPARQL queries (presented in Listing 5.2, 5.3, and 5.4) are elaborated. The concepts and the relationships between those concepts are shown in Figure 5.29.

- **owl:Thing** - This is the base class of all ontology classes.
- **skos:Concept** - A fundamental element of the SKOS ontology that allows one to assert that a resource is a concept. A relationship `R rdfs:type C`, indicates that the resource `R` is an instance of (type of) concept `C`. Some examples of concepts include “architectural patterns”, “software design patterns”, “web application framework”, and “relational database management systems”. Furthermore, each concept can be hierarchically structured using the `skos:broader` relationship. For example, the concept “patterns” is a broader concept of “software design patterns”.



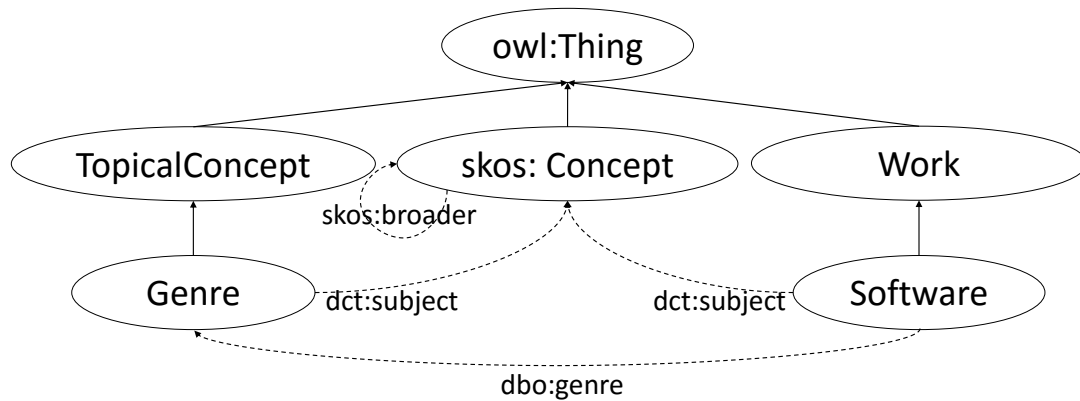


Figure 5.29: A subset of concepts in the DBpedia ontology that are relevant for the recommendation component

- **TopicalConcept** - TopicalConcept is a class defined in the DBpedia ontology. It is the base class for concepts such as “genre”, “academic subject”, “mathematical concept”, etc.
- **Genre** - Genre is a subclass of TopicalConcept and allows one to capture the genre of a specific object. Instances of Genre include, for instance, “relational database”, “graph database”, “service-oriented architecture”, etc. An instance of Genre, in our context, can be realized by multiple concrete software solutions. Hence Genre has an inverse relationship with Software through the “*is dbo:genre of*” relation. Furthermore, a Genre can belong to a higher-level Concept through the relation “*dct:subject*”. For example, Genres such as “graph database” and “column-oriented database” belong to the concept “database models”.
- **Work** - Work is a class defined in the DBpedia ontology. It captures generic properties such as *creationYear*, *developer*, and *productionCompany* which are relevant for different kinds of work including “software”, “artwork”, “article”, and “book”.
- **Software** - Software is a subclass of Work and has properties such as *description*, *release date*, *programming language*, *license*, *computing platform*, and *operating system*. As shown in Figure 5.29, the class Software has two object properties “*dbo:genre*” and “*dct:subject*”. The domain of *dbo:genre* relation is Software and its range is Genre. For instance, the software “Neo4j” which is an open-source graph database belongs to the Genre of a “graph database”. Similarly, the domain of *dct:subject* relation is Software and its range is skos:Concept. Continuing with the example of “Neo4J”, it has a *dct:subject* relation with the “NoSQL” Concept in DBpedia ontology.

The aforementioned concepts and their relationships are used for formulating the SPARQL queries and for providing recommendations to the end-users of ADeX. The SPARQL query language [107] allows querying an ontology defined in RDF, OWL, etc. For each of the annotated element in the textual description of design decisions, SPARQL queries are executed by the query executor sub-component in the Akre-Server (cf. Figure 5.28). The two specific types of recommendations a) recommending software solutions to realize an ADD and b) recommending alternative solutions related to an ADD are discussed in the subsequent sections.

**Rec 1: Recommend software solutions to realize an ADD**

To recommend software solutions, two different SPARQL queries handle the relevant types - Genre and Concept. If the annotated architectural element is an instance of Genre, then the query in Listing 5.2 is executed. The scenario with the instance of Concept also results in a similar SPARQL query and is not illustrated here. In the example below, \*variable gets replaced by the corresponding architectural element (e.g., [http://dbpedia.org/resource/Relational database](http://dbpedia.org/resource/Relational_database)). Recommendations related to the selected architectural element are presented to the end-users, for example, if a user selects “Java” in the document, alternatives to Java programming language are presented.

```

1 PREFIX dct:<http://purl.org/dc/terms/>
2 PREFIX dbo:<http://dbpedia.org/ontology/>
3 PREFIX ns:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
4 SELECT DISTINCT ?software WHERE {
5   {
6     SELECT ?software WHERE {
7       ?software dbo:genre *variable .
8       ?software ns:type dbo:Software }
9   } UNION {
10    SELECT ?software WHERE {
11      *variable dct:subject ?concept .
12      ?software dct:subject ?concept .
13      ?software ns:type dbo:Software }
14   } UNION {
15    SELECT ?software WHERE {
16      *variable dct:subject ?concept .
17      ?genre dct:subject ?concept .
18      ?genre ns:type dbo:Genre .
19      ?software dbo:genre ?genre .
20      ?software ns:type dbo:Software }
21   }
22 }

```

Listing 5.2: SPARQL query for retrieving software solutions

The query in Listing 5.2 is executed against the DBpedia ontology using the Apache Jena framework [108]. This query aggregates the results of three sub-queries wherein each subquery extracts software resources for a given Genre. For instance, the results for the Genre “Relational database” include the URIs of software such as “SQLite”, “MySQL”, “MariaDB”, “SQL Server”, etc. Each URI is further processed to retrieve the meta-information including label, description, and license which sent back to the client as an array of JSON objects.

**Sub-query 1:** Retrieves Software resources related to a Genre (for example, “Relational database”) using the *dbo:genre* relationship. The result of the Select statement is a set of URIs of the software resources (indicated by ?software).

**Sub-query 2:** This sub-query focuses on “Genre dct:subject Concept” and “Software dct:subject Concept” relationships. For a Genre (such as, “Relational database”), first, all the Concepts are retrieved and then the Software resources related to each of the Concept is extracted.

**Sub-query 3:** This sub-query ensures that the Software resources corresponding to similar Genres are extracted. For a Genre, first, its related Concepts are retrieved using the *dct:subject* relationship. Subsequently, for each of the Concept all the Genres and their corresponding Software resources are added to the result. This sub-query can also be extended based on the *skos:broader* relation for a wider set of results.

**Rec 2: Recommend alternate solutions (architecture styles, patterns, or technologies) related to an ADD**

In the previous step, the aim was to recommend software solutions for higher-level concepts (Genre). However, in this step, alternative solutions belonging to the same abstraction level (Genre or Software) are proposed. In other words, alternatives to the annotated architectural elements are extracted from the DBpedia ontology.

```

1 PREFIX dct: <http://purl.org/dc/terms/>
2 PREFIX dbo: <http://dbpedia.org/ontology/>
3 PREFIX ns: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
4 SELECT DISTINCT ?alternative WHERE {
5   {
6     SELECT ?alternative WHERE {
7       *variable ns:type dbo:Genre .
8       *variable dct:subject ?concept .
9       ?alternative dct:subject ?concept .
10      ?alternative ns:type dbo:Genre }
11   } UNION {
12     SELECT ?alternative WHERE {
13       *variable ns:type dbo:Software .
14       *variable dbo:genre ?genre .
15       *variable dct:subject ?subject .
16       ?alternative dbo:genre ?genre .
17       ?alternative dct:subject ?subject .
18       ?alternative ns:type dbo:Software }
19   }
20 }

```

Listing 5.3: SPARQL query for retrieving alternative solutions

**Sub-query 1:** If the annotated element is an instance of Genre then, the first select clause of the SPARQL query in Listing 5.3 is applicable. For instance, if the annotated element is “Relational database”, then the output includes “Key-value database”, “Column-oriented DBMS”, “Document-oriented database”, “Object database”, etc. as alternatives. Note that \*variable is replaced by the annotated element in the SPARQL query. The query first checks if the resource is of type Genre, then it retrieves all its associated Concepts using the *dct:subject* relation. Next, using the inverse relationship again it retrieves all the other Genres of the identified concepts.

**Sub-query 2:** This sub-query returns alternative solutions for concrete software solutions. It navigates the graph structure of the DBpedia ontology to retrieve software resources belonging to the same Genre. That is, if the annotated element is “MongoDB”, the result (?alternative) comprises of “CouchDB”, “OrientDB”, and “ArangoDB”.

To improve the precision of recommendations related to architectural styles and design patterns,

we composed another query with a predefined filter. The logic of this query is similar to the previous queries, that is, it involves graph navigation using the object property (*dct:subject*). The query shown in Listing 5.4, recommends alternative architectural styles and patterns. For instance, corresponding to an architectural style such as “multi-tier architecture” the retrieved results include “service-oriented architecture”, “microservices”, “entity component system”, etc.

```
1 PREFIX dct: <http://purl.org/dc/terms/>
2 SELECT DISTINCT ?alternative WHERE {
3   *variable dct:subject ?concept .
4   ?alternative dct:subject ?concept .
5   FILTER(regex(?concept, "pattern", "i"))
6 }
```

Listing 5.4: SPARQL query for retrieving alternative architectural styles and design patterns

Once the recommendations for alternatives are generated (both for Rec 1 and Rec 2), a confidence score is generated using Wiki Trends<sup>23</sup> ratings. An average of the trend score for the past five years on a monthly basis is computed and assigned as the confidence score for each of the alternative solution recommendations. A descending confidence score then sorts the results before presenting to the end-users. Moreover, users also have the option to either accept or reject recommended results corresponding to each annotated architectural element. Consequently, the confidence score is either incremented or decremented accordingly in the database. Recommendations with a confidence score below a configurable threshold value are filtered out and not shown to the users. Furthermore, as elaborated in the next subsection, the database consisting of annotated architectural elements, their corresponding recommendations, and confidence scores acts as a cache for improving the performance of the recommendation system.

### 5.5.1.4 System design of the ontology-based recommendation component

The annotation and the recommendation services are provisioned as REST services in ADeX. Once all the issues within a project have been imported using the SyncPipes component and have been labeled as either a “design decision” or “not a design decision”, the request to annotate the textual description of all design decisions is made by the Akre-Server component. As shown in Figure 5.30, the architectural elements controller handles the REST request to update all design decisions with the annotations. For each design decision, this controller invokes the document annotation functionality within the document controller. Subsequently, the document controller initializes the concept annotator UIMA pipeline and executes three annotators (cf. Section 5.5.1.1). The result is returned to the architectural elements controller for persistence.

On receiving the annotations for each of the design decision, the architectural elements controller saves the annotations within the collection named Architectural Elements (cf. Figure 5.31) within the Mongo database. Furthermore, in the background, on receiving the response from the concept annotator pipeline, the document controller also invokes the software recommendation

---

<sup>23</sup><http://www.wikipediatrends.com/>

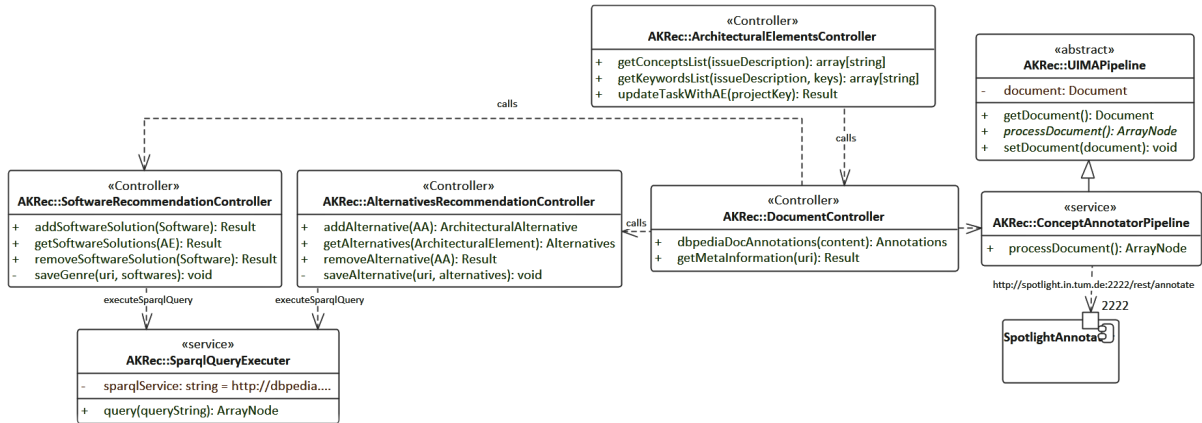


Figure 5.30: The system design of the ontology-based recommendation component

controller and the alternative recommendation controller for each of the annotations to generate recommendations Rec 1 and Rec 2 respectively. As shown in Figure 5.30, these two controllers, in turn, execute the corresponding SPARQL queries (queries presented in Listings 5.2 to 5.4) through the SPARQL query executor. On generating software solution recommendations (Rec 1), they are persisted in the SoftwareSolution collection as shown in Figure 5.31. Similarly, recommendations about alternative solutions for each of the annotated architectural element is stored in the ArchitecturalAlternative collection.

Persisting the annotations and the corresponding alternative solution improves the performance of the entire application. When the already annotated design decision needs to be presented on the UI of ADeX, the UIMA pipeline need not be executed again. The already persisted annotations and their corresponding recommendations are fetched from the persistence storage (which acts as a cache) and are presented on the UI.

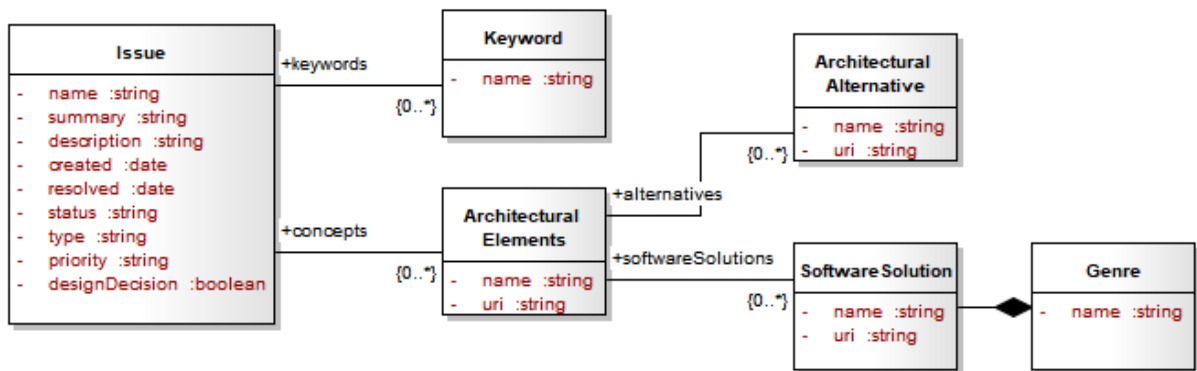


Figure 5.31: Concepts and their relationships for persisting annotations and suggestions in the recommendation component

### 5.5.2 Rationale extractor

The rationale extractor sub-component addresses the use case of identifying the rationale of a design decision. When a decision is made to address a specific quality attribute, that quality attribute is recorded as the rationale for the decision. To automatically identify the quality attribute(s) captured in the textual description of a design decision, the list of quality attributes documented in the ISO/IEC 25010 standard is used. As shown in Figure 5.32, the quality attributes of a software system are classified into design-time and run-time quality attributes. Developers need to consider the portability and maintainability of software systems as internal quality attributes which influences systems' sustainability. On the other hand, quality attributes such as usability and efficiency are referred to as external quality attributes which affect the end-uses of software systems in operational mode. Furthermore, to address a quality attribute such as portability, architects and developers need to take into account quality attributes such as adaptability, installability, and replaceability which are shown as the leaf nodes in Figure 5.32.

For each of the quality attribute, corresponding synonyms and their representing keywords are maintained in the database. These keywords were identified from previous research [46, 109, 110] related to quality requirements detection in requirement specifications. Some of the quality attributes along with their complementary keywords are shown in Table 5.1.

It should be noted that this sub-component was implemented as a prototype only to demonstrate the feasibility of the approach and for ensuring the completeness of the ADM framework. It should also be highlighted that there exist a plethora of approaches to classify non-functional requirements (NFRs) in the requirements engineering domain. The keyword-based approaches presented in [46, 111, 112] have shown that NFRs can be automatically identified in natural language text. In [46], the authors refer to keywords as “**indicator terms**”. They showed that

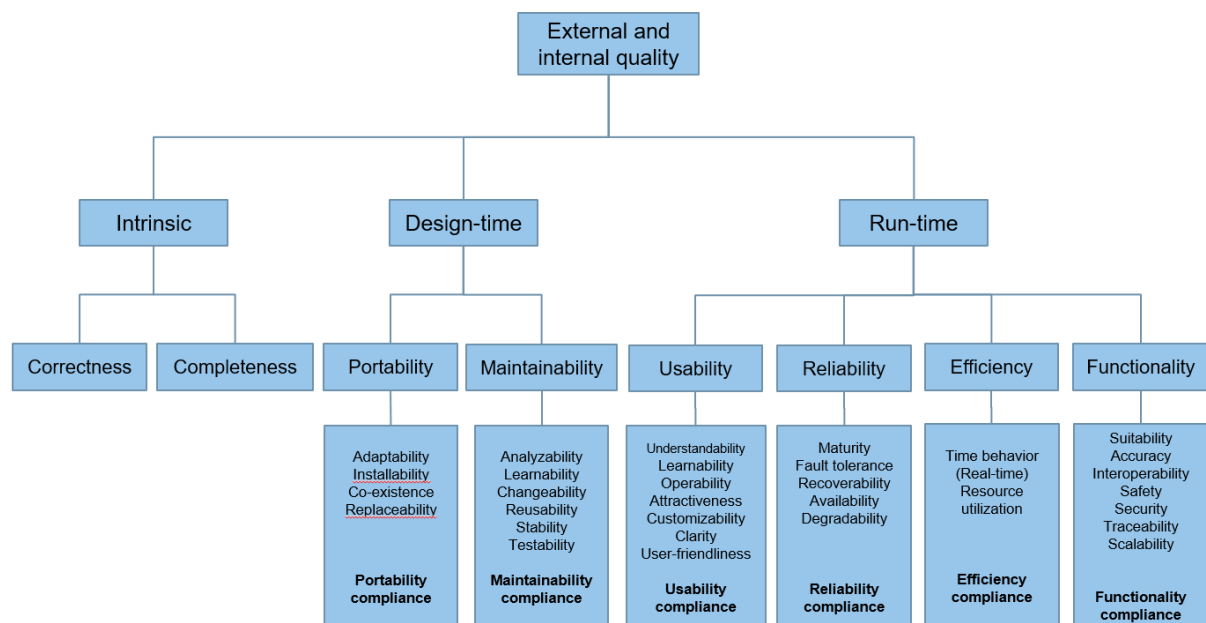


Figure 5.32: The list of quality attributes and their subcategories in the ISO/IEC 25010 standard

Quality attribute	Parent	Keywords
Portability		portable, coexistence, coexist, installability, installable
Maintainability		maintenance, stability, stable, modifiability, modifiable, modify, changable, modular, modularity, integrability, integrable
Usability		able successfully, easy, easy use, intuitive, use system, user able, usable, useful, operability, operable, learnable, learnability
Reliability		reliable, dependable, completeness, correctness, accurate, consistency, recover, recoverability, recoverable, maturity, fault tolerance, fault tolerant
Functionality		
Adaptability	Portability	flexibility, adapt, configurable, configure, customize, customizable
Reusability	Maintainability	reuse, reusable
Analyzability	Maintainability	analyze, examine, break down
Composability	Maintainability	compose
Extensibility	Maintainability	extensible
Understandability	Usability	readable, comprehensive, understandable, clear
Availability	Reliability	available, day, hour, time, year
Scalability	Functionality	simultaneous, scalable, scale, shard
Interoperability	Functionality	interoperable, interoperate
Suitability	Functionality	suitable, appropriate
Performance	Functionality	asynchronous, fast, longer, minute, multi-threading, overhead, processing, throughput, memory usage, response time, latency, execution speed, workload, time behavior, resource utilization, efficient, efficiency
Security	Functionality	access, allowed, authorized, authorized user, dispute, malicious, prevent, secure, confidentiality, authorization, encryption, password, protect, safeguard, safe, warrant, authentication
Accuracy	Functionality	accurately , accurate , exact

Table 5.1: The list of quality attributes in the ISO/IEC 25010 standard and their keywords used for automatic rationale extraction

an NFR-classifier based on a simple keyword-matching algorithm could support requirement analysts in detecting and classifying NFRs from an uncategorized requirements specification with a satisfactory recall score between 60-90% depending on the quality attribute.

The authors of [110] used a decision tree classifier equipped with a part-of-speech tagger to detect NFRs captured in software requirements specification (SRS) documents. They showed that with a 10-fold-cross-validation their approach achieves a higher accuracy of 98.56% as compared to the indicator terms based approach presented in [111] (on the same dataset). To achieve the same objective Casamayor [113], used the expectation maximization algorithm with naive Bayesian classification. The dataset used by these researchers is publicly available in the PROMISE data repository [114]. Zeng et al. [115] too used the PROMISE dataset and applied the support vector machine (SVM) algorithm with a linear kernel. They showed that for the classification of NFRs, individual words are the best index terms (as compared to N-grams and multi-word expressions) in text description of short NFRs. Zeng et al. also showed that the accuracy of classification algorithms is better on categories of large sizes than on smaller sizes. Slankas and Williams [47] in their study compared three different algorithms namely, SVM, multinomial naive Bayes, and k-nearest neighbor algorithm. They argued that, for their dataset, k-nearest neighbor performs better than multinomial naive Bayes classifier and the SVM algorithm outperforms the k-nearest neighbor algorithm. In the RE17 data challenge on the “Quality attributes (NFR)” dataset, Kurtanović and Maalej [116] demonstrated that SVM can be used to automatically classify

(with a precision and recall of 92%) requirements as functional or non-functional requirements. They further applied SVM to identify four specific NFRs: usability, security, operational, and performance. They explained that only using word features for feature selection results in higher recalls (but lower precision) for classifying NFRs than when additional syntax and meta-data features are applied.

The keyword-based approach presented in this section mimics the proposal from [46, 111]. The rationale extractor sub-component uses the quality attributes and their corresponding keywords shown in Table 5.1 and associates them with design decisions. Once all the tasks have been classified as either a design decision or not a design decision, the Akre-Server component invokes an API to update all tasks with quality attributes. In the current implementation, the textual description of a design decision is compared with the keywords of quality attributes using basic string matching process. The matched quality attributes are then persisted in the Mongo database for generating the necessary UIs.

In the context of this dissertation, tagging design decisions with quality attributes allow end-users of ADeX to get a rough estimate about how many design decisions were made to address a specific quality attribute or to investigate those quality attributes that have not been addressed during the development and maintenance of the respective software system.

### 5.5.3 Expert recommender

This sub-component realizes the use case of automatically identifying appropriate software architects and developers to support the design decision-making process. It uses the results from the previous sub-components to recommend experts who could be involved in addressing specific open design concerns.

Figure 5.33 shows the high-level overview of the approach used to implement this expert recommender sub-component. During the **training phase**, first, all the existing issues in a project are imported using the SyncPipes component. In Step 2, the decision detector component filters those issues that reflect design decisions. In Step 3, architectural elements and quality attributes within the textual description of design decisions are automatically annotated by the annotator component. For each design decision, the annotated elements, the quality attributes, and the individual who made the design decision are used to create an expertise matrix in Step 4a, which is then persisted for subsequent use.

During the **application phase** (in the lower part of Figure 5.33), a newly created issue goes through Steps 1, 2, and 3 as described before. A concept vector corresponding to the open design

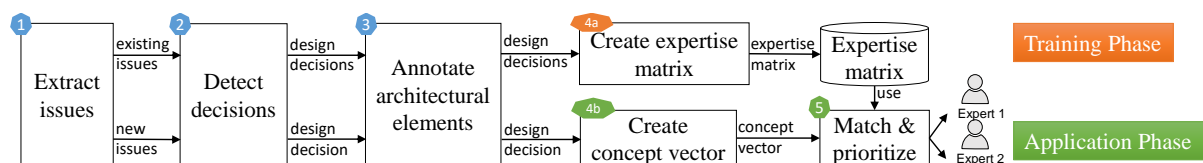


Figure 5.33: The high-level overview of the approach for recommending experts who could be involved in the ADM process



decision is computed in Step 4b. This vector is then compared against the expertise matrix in Step 5 to generate a list of experts who could discuss the corresponding concern.

### 5.5.3.1 Recommendation of experts using an expertise matrix

Since, Step 1 (extract issues), Step 2 (detect decisions), and Step 3 (annotate architectural elements) have already been discussed in Section 5.2, 5.3, and 5.5.1 respectively; this section elaborates on the remaining steps.

In Step 4(a), to automatically build the expertise matrix, the concepts from a previously proposed expert recommendation system are reused. In particular, the terms, expertise atoms and expertise matrix were introduced in [49]. With the expertise matrix, individuals' expertise profiles are represented as rows, and architectural elements are represented as columns. Let:

- $D = \{d_1, d_2, d_3, \dots, d_m\}$  be the set of architects and developers,
- $E = \{e_1, e_2, e_3, \dots, e_n\}$  be the set of architectural elements, and
- $V_{mn}$  be the expertise matrix.

Here,  $m$  is the total number of architects and developers and  $n$  is the total number of architectural elements identified by the annotator component.

**Expertise atoms** (EAs) are the elementary atoms of expertise. They reflect an individual's expertise in a specific architectural topic. Each element  $V[i][j]$  in  $V_{mn}$  represents an EA. By resolving a design concern, an individual gains expertise related to those architectural elements contained within that concern. The total count of the occurrences of an architectural element in all the design decisions resolved by an individual indicates his or her expertise level for the corresponding architectural element. That is, the higher the expertise level of an individual for an architectural element ( $V[i][j]$ ) is, the more is the expertise in handling concerns related to the corresponding architectural element. The expertise level of an individual for an architectural element can also be zero, which indicates that she has not yet resolved a design concern pertaining to that architectural element and quantifies that she has no expertise on that topic. Moreover, an individual can also have expertise in zero or more architectural elements.

Each row ( $V[i]$ ) within the expertise matrix ( $V_{mn}$ ) represents an expertise profile for individual architects and developers. An individual's name is extracted from the "assignee" attribute of an issue, which represents the person who resolved this issue and gained expertise henceforth. Using this expertise profile, one can quantitatively assess the expertise of architects and developers corresponding to each architectural element.

**Expertise profile:** Each row ( $V[i]$ ) within the expertise matrix ( $V_{mn}$ ) represents an expertise profile for individual architects and developers. An individual's name is extracted from the "assignee" attribute of an issue, which represents the person who resolved this issue and gained expertise henceforth. Using this expertise profile, one can quantitatively assess the expertise of architects and developers corresponding to each architectural element.

**Expertise matrix:** Within this matrix, rows capture expertise profiles, columns represent architectural elements, and cells correspond to expertise atoms. An excerpt of an expertise

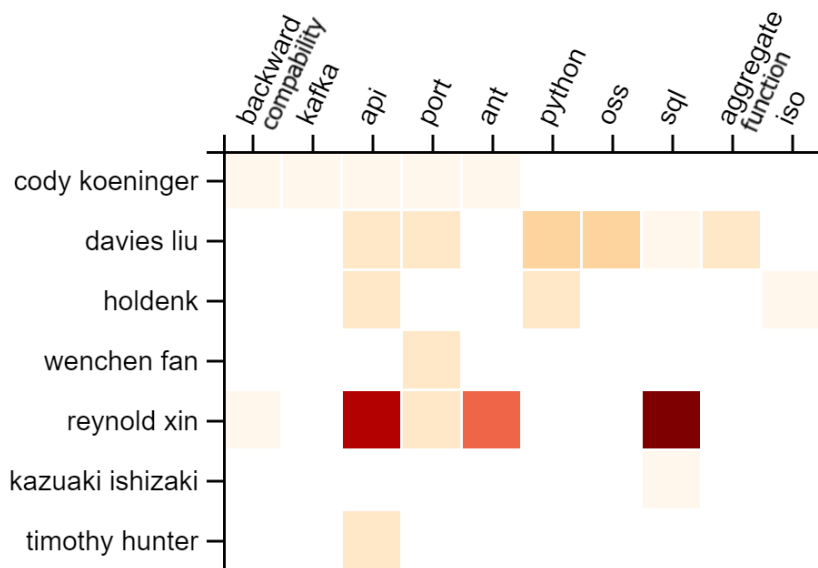


Figure 5.34: An exemplary excerpt of an expertise matrix: rows in the matrix capture the expertise profiles of individuals and columns represent architectural elements

matrix is shown in Figure 5.34. The darker the color of a cell, the higher the value for the corresponding expertise atom. Once the matrix is generated for a project, it is persisted in the Mongo database and is further used to generate a list of experts for new open design concerns.

Now, with respect to the application phase, once a new issue is added to an issue management system and imported into ADeX, the decision detector component checks if it represents a design concern. If so, the annotator component identifies the architectural elements within this new design concern. The frequency of occurrence of an architectural element within the textual description of a design concern represents its weighting factor within that concern.

In essence, first, those contributors who have expertise in one or more architectural elements within the design concern are identified and then those individuals are ranked according to their expertise level corresponding to the weighting factor of architectural elements within the new open design concern. Hence in Step 4(b), to be able to compare the textual description of a design concern against the expertise matrix, an  $n$ -dimensional sparse vector is created; where  $n$  is the total number of identified architectural elements. This vector is referred to as the concept vector (CV) of a design concern. The concept vector is represented as a one-dimensional integer array of size  $n$  and each element in the array is initialized with zero values. The position of an architectural element in the concept vector is consistent with its column in the expertise matrix. The value in the position corresponding to each architectural element present in the new design concern is then replaced by its frequency count.

- $CV = \{c_1, c_2, c_3, \dots, c_n\}$ ; where  $c_i \geq 0$ ;

The concept vector computed in the previous step is matched against the expertise matrix generated and in Step 4 (a) to generate the list of experts. The pseudocode for generating the expert list is described in Algorithm 1. The function “MATCH” takes as input the concept

**Algorithm 1** Match and prioritize

---

```

1: function MATCH(CV, Vmn, D)
2:   expertList ← {}
3:   for i in 0..m do
4:     expertVector ← newArray(n);
5:     for j in 0..n do
6:       EV[j] ← CV[j] × V[i][j]
7:     end for
8:     sum ← 0
9:     for j in 0..n do
10:      sum ← sum + EV[j] × EV[j] ▷ Compute score as vector magnitude
11:    end for
12:    score ← SQRT(sum)
13:    if score > 0 then
14:      expertList.add("person", D[i])
15:      expertList.add("score", score)
16:    end if
17:  end for
18:  expertList ← ORDERBY(expertList, "score")
19: end function

```

---

vector, expertise matrix, and the set of architects and developers. For each expertise profile (row) in the matrix, an expert vector ( $EV$ ) of size  $n$  is created. Each element in  $EV$  is the product of the frequency of an architectural element ( $CV[j]$ ) in a new design concern and the expertise level ( $V[i][j]$ ) of the respective individual corresponding to that architectural element. For instance, if a new design concern emphasizes an architectural element with a higher frequency count and a specific individual has more expertise with that architectural element, then the score for that individual should proportionally increase with respect to both these variables. Once the expert vector is generated for an individual, expertise  $score$  is calculated as the magnitude (vector length) of that expert vector. The magnitude of the expert vector is calculated as the square root of the dot product of the vector by itself. Hence, the expertise score generated for an individual is equally distributed across all architectural elements in the new design concern. If this score is greater than zero, the corresponding individual along with the expertise score is added to the expert list. As a last step, after iterating over all expertise profiles, the expert list is ordered by the expertise score.

#### 5.5.4 System design of the Akre-Server component

As discussed in the introduction of this Chapter, the Akre-Server component also acts as a middleware that coordinates the tasks related to preprocessing all issues within a project. This component interacts with the other components namely, Syncpipes, Document classifier, and Document clustering and invokes the corresponding services within those components.

The workflow of preprocessing all the issues within a project is as follows:

1. Once a request to create a new project is made by the Amelie - Decision explorer client, the Akre-Server invokes the SyncPipes component (cf. Section 5.2) to import all the issues within that project into the issues collection in the Mongo database.
2. On importing all the issues from Jira, as shown in Figure 5.35, the preprocess controller calls the label design decision controller. This controller delegates the task to the Document

## 5. System design and implementation

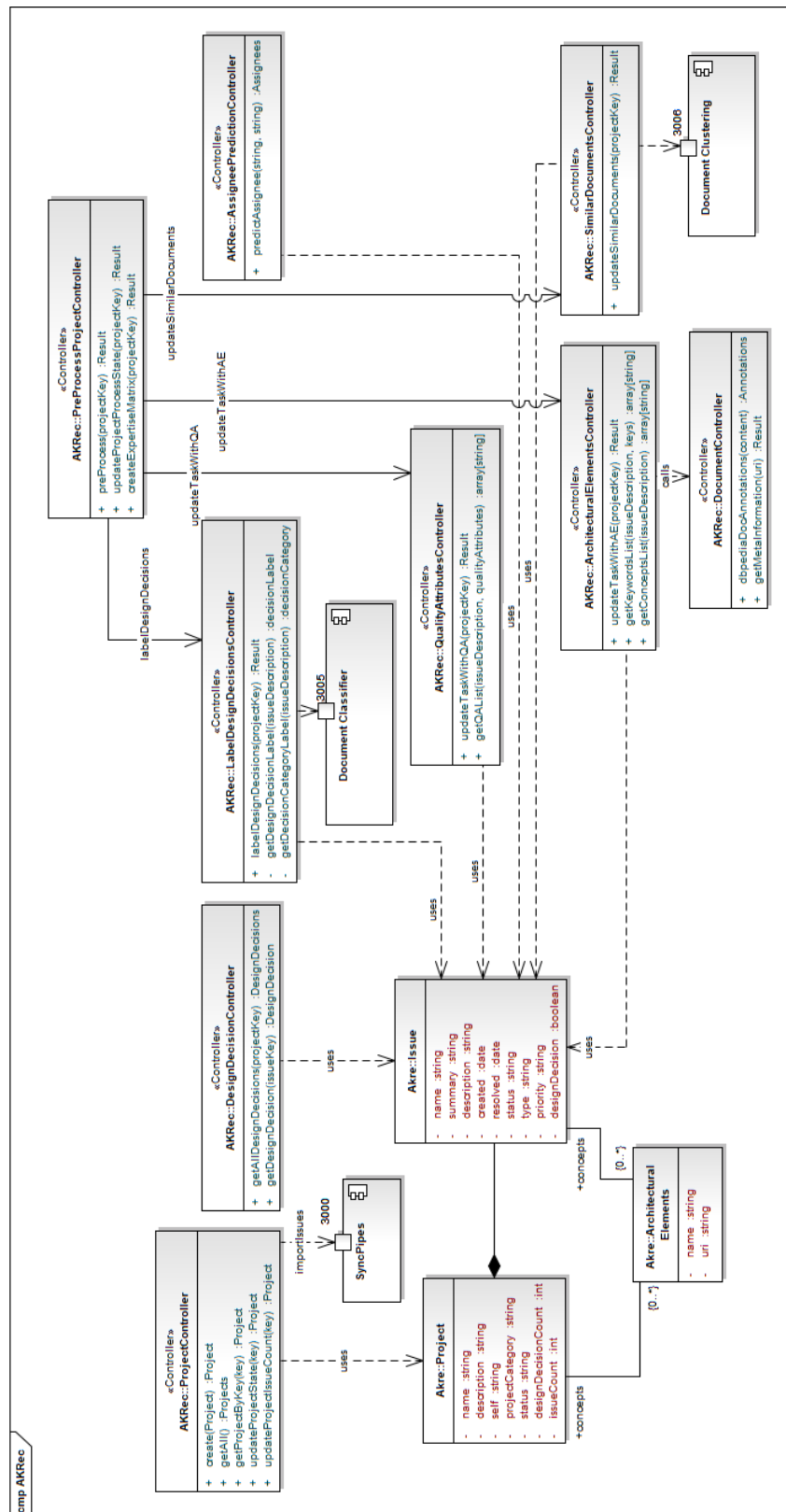


Figure 5.35: The system design of the Akre-server component: the design shows the application controllers and their dependencies on the data models

classifier component (cf. Section 5.3) to label all design decisions and to further classify them into different decision categories.

3. Once all the design decisions are identified within the project, the preprocess controller invokes the task of associating all design decisions with quality attributes (as discussed in subsection 5.5.2).
4. Next, the preprocessing step involves the task of annotating all design decisions with their respective architectural elements (cf. subsection 5.5.1).
5. On annotating all the design decisions with architectural elements and quality attributes, next, the task to create an expertise matrix is invoked.
6. Finally, the preprocess controller invokes the service provided by the Document clustering component (cf. Section 5.4) to create and save the cluster model capturing similar design decisions within those clusters.

Apart from preprocessing issues within a project, this component also facilitates the Amelie - Decision explorer client with the necessary business logic. For example, the assignee prediction controller uses the expertise matrix to recommend experts who could be involved in addressing design concerns. Furthermore, this component also provides all the necessary data for creating various visualizations (as presented in the next section) to the decision explorer client.

## 5.6 Amelie - Decision explorer client

The Amelie - Decision explorer client<sup>24</sup> is the main front-end of ADeX. This front-end is an independent component that interacts with the Akre-Server component and provides the necessary visualizations to the end-users of ADeX. As the name of the component suggests, end-users of ADeX can examine and analyze those design decisions made in large (legacy) software projects from different perspectives.

This component is built using node.js<sup>25</sup> and relies on a JavaScript library called React<sup>26</sup> for its UI components. All the graphical visualizations are implemented using another JavaScript library called D3.js<sup>27</sup>. The data necessary for generating the UIs is fetched from the Akre-Server over the predefined REST endpoints.

In this section, an effort is made to explain how end-users can effectively use ADeX by elaborating on each of the UI within the Amelie - Decision explorer client.

The first view a user encounters is the data table of a list of projects (only their name and some meta-information) that have already been imported from Jira. On this view, the user can search for a project's name, keywords within its description, or filter projects based on project categories. Furthermore, if issues within a project have already been imported through the SyncPipes component, then the user sees the total number of issues within that project in the

---

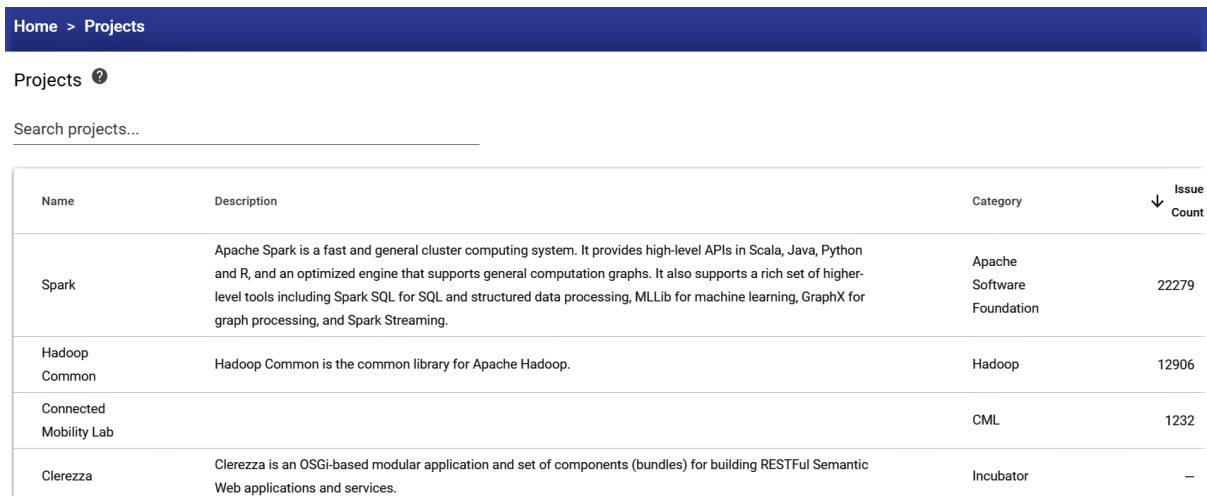
<sup>24</sup><https://amelietor-9f8c3.firebaseio.com>

<sup>25</sup><https://nodejs.org>

<sup>26</sup><https://reactjs.org>

<sup>27</sup><https://d3js.org/>

## 5. System design and implementation



Name	Description	Category	Issue Count
Spark	Apache Spark is a fast and general cluster computing system. It provides high-level APIs in Scala, Java, Python and R, and an optimized engine that supports general computation graphs. It also supports a rich set of higher-level tools including Spark SQL for SQL and structured data processing, MLlib for machine learning, GraphX for graph processing, and Spark Streaming.	Apache Software Foundation	22279
Hadoop Common	Hadoop Common is the common library for Apache Hadoop.	Hadoop	12906
Connected Mobility Lab		CML	1232
Clerezza	Clerezza is an OSGi-based modular application and set of components (bundles) for building RESTful Semantic Web applications and services.	Incubator	–

Figure 5.36: A datatable in ADeX shows the list of projects that have been imported from Jira

last column. For instance, as shown in Figure 5.36, the Apache Spark<sup>28</sup> project has already been imported and contains 22,279 issues. On the other hand, the issue count for the project named Clerezza<sup>29</sup> is empty, indicating that the project has not been imported into ADeX.

If a user wishes to import and analyze design decisions in a project that has not yet been imported, selecting the respective project in the data table takes the user to the UI shown in Figure 5.37a. Through this view, first, the user can click on the Import button to import all the issues from Jira into ADeX. This action causes the execution of a pipeline within the SyncPipes component as explained in Section 5.2. Once all the issues are imported, next, the user can click on the Process button as shown in Figure 5.37b. This action executes a series of tasks to extract architectural relevant information from issues. In particular, first all the design decision are labeled, they are classified into different decision categories, quality attributes and architectural elements are annotated within the automatically labeled design decisions, expertise matrix is generated, and finally, cluster model containing similar design decisions are created. The execution of these steps is handled by the Akre-Server component as explained in Section 5.5.4. Once all the design decisions are processed, the user is directed to the web page shown in Figure 5.38. Also, if a project was already processed, clicking on the project in the data table (cf. Figure 5.36) also directs the user to this web page.

In this view, on the top panel, the user gets an overview of the selected project. It includes project description retrieved from Jira, the total number of issues related to that project, and the number of identified design decisions. As shown in Figure 5.38, Apache Spark contains 22,279 issues which were extracted from Jira and 463 issues which were labeled as design decisions.

The view on the web page shown in Figure 5.38, corresponds to the quality attributes view (first tab). This view helps the user analyze how many design decisions were made to address specific quality concerns. The design decisions are also categorized (using color codes) according to the

<sup>28</sup><https://issues.apache.org/jira/projects/SPARK>

<sup>29</sup><https://issues.apache.org/jira/projects/CLEREZZA>

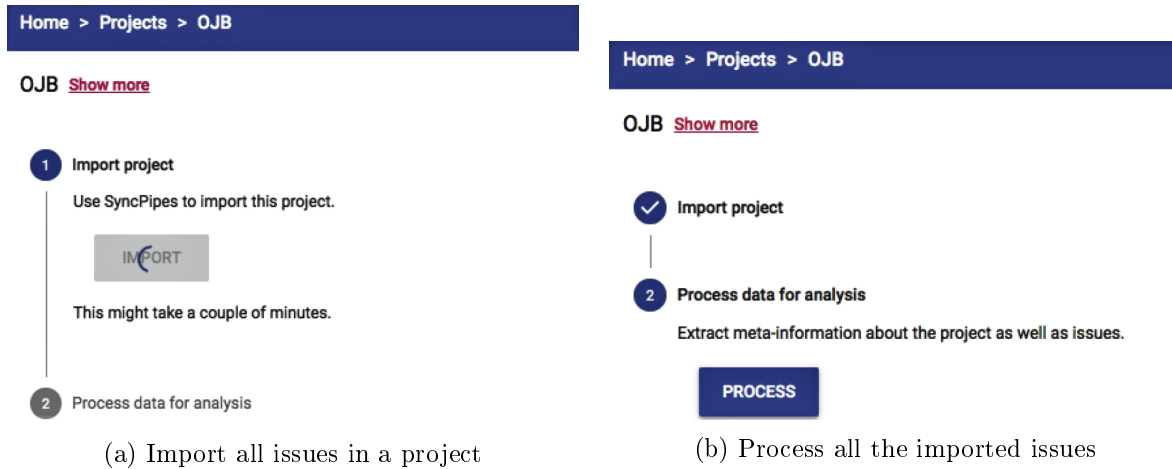


Figure 5.37: User interfaces to import all the Jira issues of a project and to execute the processing pipeline for decision detection and annotation

decision categories, namely, structural, behavioral, and non-existence/ban decisions. The structural decisions are indicated with dark blue color, behavioral and ban decisions are highlighted using light blue and orange colors respectively. On the right side of the panel, those quality attributes corresponding to which no design decisions were made are listed. This helps users to quickly analyze those quality attributes that needs more attention.

For instance, corresponding to 462 design decisions in the Apache Spark project, 27 design decisions were made to address performance related concerns. Within those 27 design decisions, there are 3 structural decisions, 19 behavioral, and 5 non-existence/ban design decisions. Furthermore, no design decisions correlated to composability, portability, extensibility, and interoperability. Also, given the fact that Apache Spark is an analytical engine for processing big data, understandably, most of the design decisions are related to performance, availability, and security. Only a few decisions were made concerning suitability, reusability, and adaptability.

The user can click on any of the segment within the bar chart to view the list of corresponding design decisions in a data table. For instance, clicking on the orange segment on the bar related to performance shows only those five non-existence/ban design decisions in a data table (as shown in Figure 5.42). This allows the end-user to drill-down from 22,279 issues to only those specific types of design decisions, and analyze them further. This facilitates the idea of narrowing the view of an architect to facilitate him/her to focus only on those aspects that are interesting from an architectural perspective.

The tab adjacent to Quality Attributes is related to Architectural elements. Clicking on this tab shows a bubble chart containing architectural elements which were discussed within the project's design decisions. As shown in Figure 5.39, each bubble within the bubble chart corresponds to an architectural element. And the size of a bubble reflects the frequency of the occurrence of that architectural element (also indicated with the frequency count) in all design decisions of the project. These architectural elements are automatically identified by the architectural elements annotator component using the DBpedia ontology (cf. the discussion in Section 5.5.1). Similar

## 5. System design and implementation

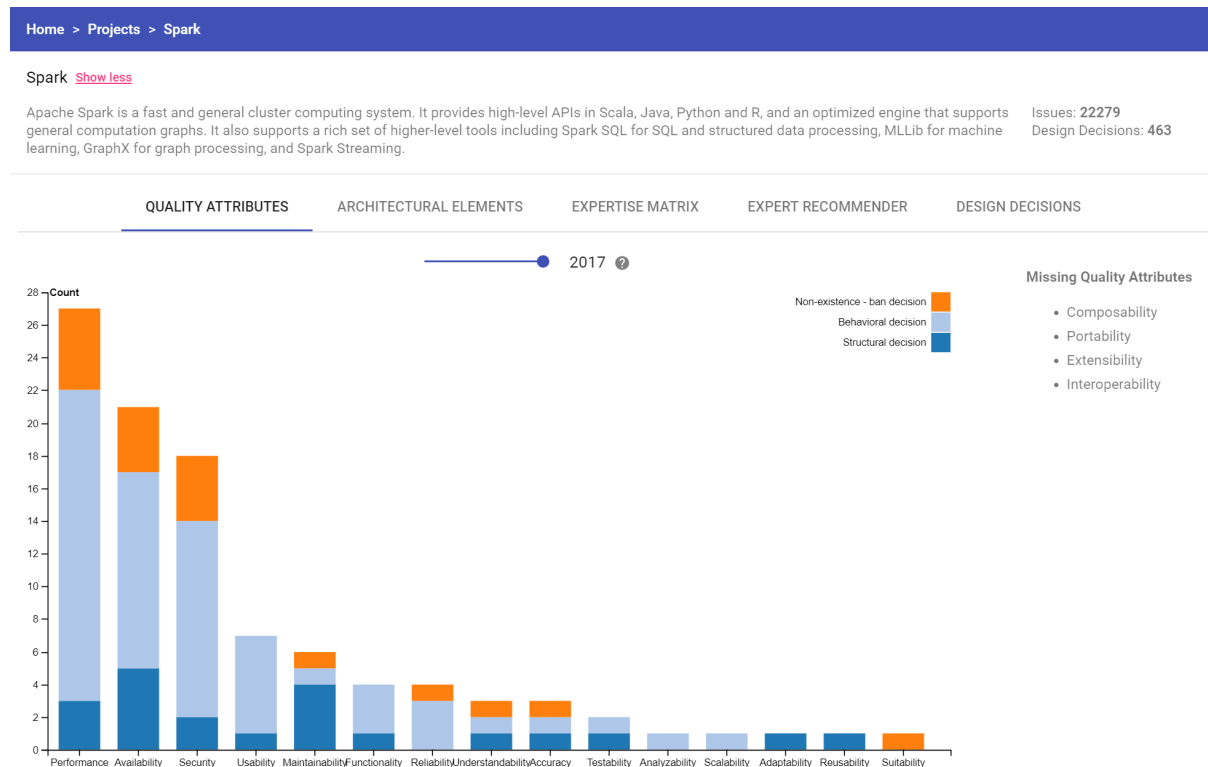


Figure 5.38: Quality attributes viewpoint: each individual bar represents a quality attribute and the segments within a bar indicate the design decision category

to the quality attributes view, the scroll bar on the top allows a user to view the evolution of architectural elements over time. That is, sliding the scroll bar to the year 2015, for instance, shows those architectural elements that were relevant at that time. The evolutionary view helps a user analyze which architectural elements were newly introduced and their influence over a period of time. Finally, clicking on a specific architectural element takes the user to the data table containing design decisions related to that specific architectural element.

The bubble chart in Figure 5.39, pertains to design decisions in the Apache Spark project. Since Apache Spark is a data processing engine, which exposes its functionalities over well-defined interfaces, there were many (70) design decisions concerning APIs. Furthermore, since Apache Spark supports different programming languages, they are indicated with the bubbles of Java, Python, and Scala. The view also shows that there were design decisions made related to a communication protocol such as HTTP and HTTPS. Clicking on the HTTP bubble will display only 25 design decisions out of 22,279 issues to the user (reflecting the idea of allowing users to narrow down the large search space to only those interesting aspects).

The next tab in the list, after Architectural elements is the Expertise matrix. Clicking on this tab takes the user to the interface shown in Figure 5.40. This UI presents the expertise matrix; where columns represent architectural elements and rows correspond to the contributors within a project. The cells in the expertise matrix are color coded. The darker a cell, the more exper-





## 5. System design and implementation



Figure 5.40: Expertise matrix viewpoint: rows correspond to individuals, columns capture architectural elements, and cells indicate the expertise of an individual on the corresponding architectural element

open design concern. By default, only two experts most suited for solving the design concern is shown to the user. However, clicking on the down arrow, as shown in Figure 5.41, shows all the experts with their expertise score. The list of experts is ordered according to descending expertise score. The higher expertise score for an individual indicates that she is more suited to address the respective concern. The experts for addressing an open design concern are identified using the algorithm presented in Listing 1 of Section 5.5.3.

Based on this idea of recommending experts to address design concerns, a Master's thesis was conducted by Kevin Koch [117]. In this Master's thesis, a (richer) sophisticated UI was developed. It included features to add/remove experts from the recommendations manually. Predefined teams of experts could be assigned to resolve certain design concerns. It also contained a clear separation of roles (moderator, a team of architects and developers for resolving a design concern, and decision owner) within the expert recommender system. Furthermore, the UI also provided options to include novices along with experts in a decision-making team to ensure knowledge exchange among team members. This Master thesis was conducted as part of an extension to the work presented in Section 5.5.3 (cf. [117] for a detailed report).

The last tab in the list corresponds to all the design decisions in a project. As shown in Figure 5.42, all the design decisions within a project are listed using a data table. The data table is integrated with search and filtering options. The end-user can perform a full-text search on the title and description of design decisions or filter only those design decisions corresponding to a specific quality attribute or architectural element. The data table captures the ID of design de-

Figure 5.41: A datatable showing the recommendation of experts and their expertise score for addressing open design concerns

isions as well as the title and description, which are retrieved from Jira. The quality attributes column shows the associated quality attributes that are tagged by the rationale extractor component and the architectural elements column contains those elements that are annotated by the architectural elements annotator component. The last column presents the category of design decisions (structural, behavioral, and ban/non-existence decision). Each design decision (row) in the data table is selectable and clicking on a design decision allows a user to analyze that decision through the UI shown in Figure 5.43.

This last view on the design decision is a detailed view that facilitates architects and developers to make informed design decisions. As shown in Figure 5.43, the left section of the UI shows the textual description (title and description) of the selected design decision. On the lower

ID	Design Decision	Description	Quality Attributes	Architectural Elements	Decision Category
12177	Update KafkaDStreams to new Kafka 0.10 Consumer API	Kafka 0.9 already released and it introduce new consumer API that not compatible with old one. So, I added new consumer api. I made separate classes i ...		Apache Spark, backward compatibility, kafka, API, Apache	Structural decision
13671	Use different physical plan for existing RDD and data sources	Right now, we use PhysicalRDD for both existing RDD and data sources, they are becoming much different, we should use different physical plans for the ...			Structural decision
9774	Add Python API for ml.regession.IsotonicRegression	Add Python API user guide and example for ml.regession.IsotonicRegression		ISO, Python, GUI, API, IDE	Behavioral decision

Figure 5.42: A datatable showing the list of automatically detected design decisions, their associated quality attributes and architectural elements

Home > Projects > Spark > SPARK-12177

### Update KafkaDStreams to new Kafka 0.10 Consumer API

Kafka 0.9 already released and it introduce new consumer API that not compatible with old one. So, I added new consumer api. I made separate classes in package `org.apache.spark.streaming.kafka.v09` with changed API. I didn't remove old classes for more backward compatibility. User will not need to change his old spark applications when he uprgade to new Spark version. Please review my changes

Annotating...

ANNOTATE

Design Decision	Summary
SPARK-9769	Python API for ml feature CountVectorizer
SPARK-9774	Python API for ml regression IsotonicRegression
SPARK-12177	KafkaDStreams to new Kafka 0 10 Consumer API

Figure 5.43: Recommendations related to a design decision: annotated architectural elements, their alternatives, and similar design decisions made in the past

part of the screen, a button to annotate the textual description is available. Clicking on this annotate button invokes a service in the Akre-Server component to process the design decision and to annotate it with architectural elements using the DBpedia ontology (as discussed in Section 5.5.1). This results in highlighting the textual description with architectural elements. Next, when the end-user selects an architectural element, an info-box with three tabs is shown on the right side of the screen. The first tab corresponds to the meta-information about the selected architectural element which is retrieved from the DBpedia ontology. It includes a description of the architectural element, the type of license (open-source, closed-source), external URL, etc. The second tab corresponds to the recommendations about the alternative solutions at the same granularity. For example, if the architectural element “relational database” is selected, the recommendations in this tab could include “document-oriented database”, “graph database”, and “key-value store”. These recommendations are generated using the SPARQL query presented in Listing 5.3. The third tab corresponds to the recommendation of alternative software solutions to realize a decision. For example, for the same architectural element “relational database”, the recommendation includes “MySQL”, “Microsoft SQL”, and “SqlLite”. As already discussed in Section 5.5.1, these recommendations are generated by executing the SPARQL query presented in Listing 5.2. End-users also have options to add new or delete recommendations through the UI. These recommendations will then be either shown or removed in the next iteration.

Lastly, an info-box is shown on the right side of the screen. It presents the ID, summary, and similarity scores (cosine and Jaccard scores) of decisions similar to the selected decision. These scores are generated based on the explanation provided in Section 5.4. End-users can select a similar design decision and analyze it to understand how it was resolved.

This chapter presents the evaluation of different components within the ADeX system. The evaluation is categorized into two sections: quantitative and qualitative evaluation. In quantitative evaluation, the accuracy of the approaches within specific components are measured and documented. Whereas in qualitative evaluation, feedback gathered during presentations and informal interviews with the project partners in an industrial setting is documented. The quantitative evaluation helps to measure the correctness of the presented approaches. Whereas, the quantitative evaluation provides insights on the completeness and applicability of ADeX in real-world projects.

## 6.1 Quantitative evaluation of components within ADeX

In this section, the quantitative evaluation of the following components is presented:

1. Architectural elements annotator and alternative solutions recommender component
2. Decision detection and classification component
3. Expert recommendation component

It should be noted that the components, namely, the document clustering (for finding similar design decisions) and the rationale extractor (for mapping quality attributes with design decisions) were implemented and evaluated as part of two Master's thesis projects. Hence, the evaluation corresponding to those two components can be found in [118] and [119] respectively. Furthermore, with respect to the expert recommender component, an extended evaluation of specific ML algorithms (which are not performed by this author) is documented in the Master's thesis of Koch [117].

### 6.1.1 Quantitative evaluation of the architectural annotator component

To evaluate the cross-domain ontology-based approach for annotating architectural elements within architectural description and for the recommendation of alternative solutions for ADM, an empirical study was conducted. For this study, we had set forth the following three hypothesis:

1. The DBpedia ontology can be used to extract architectural elements in text.
2. Software solutions to realize an ADD can be extracted from the DBpedia ontology.
3. Alternatives for architectural elements can be extracted from the DBpedia ontology.

As part of the evaluation setup, four chapters with the headings “Technical/IT infrastructure” and “Technical decisions” from four different software architecture documents were analyzed. These documents were aligned to a company-specific blueprint and were produced as part of large software engineering projects in the data analytics domain. All the four documents were used by our industry partner for internal and external communication. These documents were maintained by four different sub-teams of architects (7-10 people) and they used these architectural documents for communicating with stakeholders of the projects (more than 100 people).

The evaluation was conducted in two phases. The first phase covered the approach of annotating architectural elements in textual description and the second phase corresponded to the recommendation of alternative solutions.

#### *Phase 1: Evaluation of architectural elements annotator*

Two software architects (including the author of this dissertation) with more than five years of experience manually annotated the architecture documents with architectural elements. Each architect independently color-coded all four documents. After that, in a shared meeting, the independently color-coded annotations were merged. Annotations that were not accepted by both the architects during the meeting were excluded and the final number of manual annotations in each of the document were counted. Note that, irrespective of an architectural element’s occurrence frequency, an annotated architectural element is only counted once within a document. As shown in Table 6.1, document 1 contained 29 unique, manually annotated architectural elements, document 2, 3, and 4 contained 39, 31, and 49 annotations, respectively.

To get the count of automatic annotations, the same four software architecture documents were subsequently uploaded to ADeX and the input text was annotated by the DBpedia annotator

Documents	(#) Manual annotations	(#) Automatic annotations	(#) Irrelevant annotations	(%) Precision	(%) Recall	F-score
Doc 1	29	24	2	91.66	75.86	0.83
Doc 2	39	35	3	91.42	82.05	0.86
Doc 3	31	33	8	75.75	80.64	0.78
Doc 4	49	47	4	91.49	87.75	0.84

Table 6.1: Evaluation results of the automatic annotation of architectural elements in natural (English) language text

component (as discussed in Section 5.5.1.1). Some of the annotated architectural elements included “N-tier”, “client/server”, “middleware”, “Java”, “C Sharp”, “Hadoop”, and “MySQL”.

Table 6.1 shows the comparison of results between the manual and the automatic annotation of architectural elements in the architecture documents. Document 1 was automatically annotated with 24 architectural elements, document 2, 3, and 4 were annotated with 35, 33, and 47 architectural elements, respectively. Overall, 148 non-repetitive architectural elements were manually color-coded by the architects. In comparison, the DBpedia annotator automatically annotated 139 unique architectural elements. The overall precision (fraction of automatically retrieved architectural elements that are relevant) of the annotator component is 87.58%, the recall (fraction of relevant architectural elements that were successfully retrieved) is 81.57%, and the F-score (harmonic mean of the precision and recall [120]) is 0.84.

Table 6.1 shows the precision, recall, and F-score for each of the analyzed architecture document. The results for Doc 3 are least accurate (*precision* - 75.75%, *recall* - 80.64%, and *F-score* - 0.78). The reason for lower accuracy for Doc 3 was the fact that it contained organization-specific terms and acronyms used in different context. Moreover, since the document contained architectural terms used specifically within the organization (e.g., custom communication protocol, internal software system names) the recall was also low (80.64 %).

However, the results based on the remaining documents indicate that *the general architectural elements can be extracted from documents using the publicly available DBpedia ontology (hypothesis 1)*. Subsequently, other useful meta-information related to the annotated elements (including the description and the link to the Wikipedia article) can also be extracted and presented to architects during the software architecture documentation process.

### ***Phase 2: Evaluation of the recommendation system***

The two architects mentioned before manually evaluated the results corresponding to the recommendation of software solutions (Rec I) and alternate architectural solutions (Rec II). The evaluation was carried out using the UI presented in Section 5.6. For the evaluation, only the top ten recommendations were considered. This restriction was applied not only due to the impracticability of evaluating all the results in a considerable amount of time but also due to the lack of user interest in the hits positioned below the tenth position (cf. [121]). Furthermore, since it is not feasible to enumerate all relevant recommendations for each of the annotated element, the recall score was ignored and only the precision at ten (P@10) was considered. The results of both recommendations are shown in Table 6.2.

To reiterate, Rec I corresponds to the recommendation of concrete software solutions to realize a design decision. That is, for high-level architectural elements that have `rdf:Type` as `Genre` or `Concept` in the DBpedia ontology, Rec I suggests software solutions. Some of these high-

	Correct recommendations	Total recommendations	(%) Precision
Rec I	228	283	80.56
Rec II	306	379	80.73

Table 6.2: Evaluation results of the recommendation of software and alternative solutions

## 6. Evaluation

Annotated concept	Software solutions
Web container	Apache Tomcat, Apache Geronimo, Enhydra, GlassFish, JBoss Application Server, Jetty
File system	IBM General Parallel File System, Extended file system, File Allocation Table, Amazon S3
Object-oriented	Smalltalk, Ruby, Java, C++, C Sharp, Visual Basic.NET, Objective-C, TypeScript, Lava
Relational database	PointBase, TimesTen, FrontBase, DatabaseSpy, MaxDB, MySQL, MariaDB, WebScaleSQL
NoSQL	EXist, BaseX, Neo4j, Elliptics, LevelDB, FoundationDB, DocumentDB, C-treeACE

Table 6.3: Exemplary recommendations for software solutions that are automatically generated by the recommendation component

level architectural elements include “Web container”, “File system”, and “Relational database”. Table 6.3 shows a subset of the recommended software solutions for realizing some of the ADDs. For instance, while considering a web container, an architect or a developer can investigate software solutions such as Apache Tomcat, Glassfish, or JBoss application server.

Out of 139 automatically annotated architectural elements, 29 architectural elements were of type Genre or Concept. For those 29 architectural elements, a total of 283 software solutions were recommended. Among them, 228 recommendations were marked as correct by the architects. The precision of Rec I is 80.56%. *This result positively supports the hypothesis that software solutions to realize an ADD can be extracted from the DBpedia ontology (hypothesis 2).*

For Rec II, all the annotated architectural elements were analyzed. These elements, for instance, included “Apache Tomcat”, “Java”, “PostgreSQL”, and “Maven”. Some of these recommendations are shown in Table 6.4. For instance, an alternative to the N-tier architectural style is the use of service-oriented architectures; some alternatives for the Hadoop file system could be Google file system, Amazon S3, Microsoft Azure’s file system, etc. For all the annotated architectural elements, as shown in Table 6.2, a total of 379 alternatives were generated by the system. Out of which, 306 recommendations were considered as relevant by the experts. Hence, the precision of Rec II is 80.73%. *The result indicates that the alternatives for architectural elements can be extracted from the DBpedia ontology (hypothesis 3).*

During the evaluation, it was observed that, as compared to the annotated architectural elements related to architectural styles or design patterns, the recommendations related to elements representing software systems were better. This is mainly because software technologies are considerably well maintained in the DBpedia ontology as compared to some of the higher-level concepts. Notably, in the DBpedia ontology, concepts such as “Object-oriented”, “Hypertext Transfer Protocol”, or “Unified Modeling Language” belong (*rdf:Type*) to the concept *Thing*, which is a broad concept. Furthermore, these concepts also inherit properties from other broad concepts through the *dct:subject* relationship. Hence, the results related to these elements are typically incorrect.

Annotated concept	Alternative solutions
N-tier	Service-oriented architecture
COBRA	D-Bus, Internet Communications Engine, DCOM, XPCOM, IBM System Object Model
HDFS	Google File System, Ceph, Amazon S3, NFS, Microsoft Azure’s file system
Teradata	Salesforce.com, Oracle Exadata, MonetDB, HPCC, Scriptella, Sybase IQ, Apatar
JUnit	CsUnit, TestNG, Selenium, Mockito, JTriger
Model-view-controller	Pipeline, Presentation-abstraction-control

Table 6.4: Exemplary recommendations for alternative solutions that are automatically generated by the recommendation component



Due to such scenarios, it becomes necessary to limit the results of the SPARQL queries to avoid performance issues in the recommendation system. However, the drawback of enforcing such a limitation is a decreased accuracy of the recommendations. Hence, even though the results are satisfactory for architectural elements that are related to software type, *further investigation is required to extract recommendations related to architectural styles and patterns.*

### ***Limitations and threats to validity***

In this study, we identified four major limitations of the ontology-based approach and subsequently, proposed the possible solutions to addresses those limitations:

**Limitation 1:** In the ontology-based approach, we did not consider the context for the recommendations. The context must be identified based on how users interact with the system. If the user accepts an annotated architectural element, then this confirmation must be used for the recommendations that follow within the same document. For instance, if a user up-votes the term “Java” then the context of the programming language must be set a priori. That is, if the term “JUnit” follows within the document, then the alternative software solutions should correspond to “Java”. This can be accomplished by exploring the *dbo:programmingLanguage* relationship between “work” and “programming language” concepts in the DBpedia ontology.

**Limitation 2:** The relationships indicated by the dotted arrows in Figure 5.29 represent many-to-many relationships. In the SPARQL queries, we did not ignore the concepts that could lead to false positives. Prioritizing, as well as, removing some of the related concepts before executing the SPARQL queries will improve the recommendations. By identifying semantic similarity of concepts in the domain and range of a relationship, it is possible to provide a ranking mechanism to investigate the associated concepts.

**Limitation 3:** The accuracy of the recommendations relies on the DBpedia ontology. For instance, it is possible that a newly introduced software solution is not updated in the DBpedia ontology and hence, it is not reflected in the recommendations. A feedback loop from the users that directly adds new concepts to the DBpedia ontology should address this limitation.

**Limitation 4:** For this evaluation, only four software architecture documents were considered. We realize that the corpus is not large enough and the recommendation system needs to be evaluated with a larger number of software architecture documents as part of the future work.

### **6.1.2 Quantitative evaluation of the decision classifier component**

To evaluate the ML-based approach to (a) automatically detect and (b) classify the extracted decisions into three decision categories, issues maintained in two large open-source projects were used. The two projects are namely, *Apache Spark* and *Apache Hadoop Common*.

Apache Spark is a large-scale data processing engine. Since early 2014, contributors of this project have captured more than 19,000 publicly accessible issues in JIRA from version 0.9.0 to 2.1.0<sup>1</sup>. Apache Hadoop, on the other hand, is a distributed computing software and the Hadoop Common component is the core that provides utilities to the other Hadoop components such

---

<sup>1</sup>as of 25.01.2017, when the issues were imported

as YARN and MapReduce. Since early 2013, contributors of Hadoop Common project have maintained more than 10,000 issues from versions 0.2.0 to 3.0.0-alpha<sup>2</sup>. Both these projects are related to each other, as Apache Spark runs in Hadoop clusters. These two projects were selected for evaluation due to the following reasons:

- Industry partner's interest to analyze design decisions for building a data analytics platform
- Experts responsible for generating the training dataset for ML had used either one of the systems and were involved in data analytics projects
- Both are long-running projects and have maintained more than 10,000 issues
- Both these projects are extensively used in data management solutions<sup>3</sup>

For evaluating the supervised ML-based approach, first, we created a labeled dataset with training and testing data. Thereafter, different supervised ML algorithms were tested and the accuracy of the algorithms was analyzed. In particular, the following steps were carried out:

- Data extraction
- Data curation
- Manual labeling for generating the dataset
- Setting up the ML pipeline
- Analyzing the results of each of the algorithm

These steps are elaborated in the subsequent subsections.

### 6.1.2.1 Data extraction

The issues maintained in Apache Spark and Apache Hadoop Common projects were first extracted from Jira into ADeX's database<sup>4</sup>. During the extraction process, issues were extracted from Jira while filtering for the following relevant settings. The list of prerequisites for issues to qualify for the study helped to narrow down the large number of issues to those issues that potentially reflect design decisions. For instance, a *critical* task that has been *resolved* by implementation indicates that there is a potential change in the detailed design of a system.

- Issue Type = Task, New Feature, Improvement, or Epic
- Priority = Blocker, Critical, or Major
- Status = Resolved
- Resolution = Fixed, Implemented, Done, or Resolved

Issues from Jira were extracted using SyncPipes. In total, **2,259** issues from Apache Spark and **420** issues from Hadoop Common projects were extracted and persisted in ADeX.

---

<sup>2</sup>as of 25.01.2017, when the issues were imported

<sup>3</sup><https://www.gartner.com/doc/3371732/critical-capabilities-data-warehouse-data>

<sup>4</sup><https://server.sociocortex.com/typeDefinitions/1vk4hqzziw3jp/Task>

### 6.1.2.2 Data curation

As part of the description of an issue, the summary (title) and description attributes of an issue were considered. The summary and description attributes elaborately describe an issue’s purpose. It should be noted that comments within issues could also be analyzed in this context.

As a first step, the summary and description of all the extracted issues were cleaned by removing the following:

- Code snippets within the text, as well as code inside `{ }` and `{code}` blocks
- Comments inside `noformat` blocks
- URLs inside the text

The above restrictions were introduced so as to ensure that the intent of the issue could be justified only on the basis of the description without the need for code snippets for explanation.

### 6.1.2.3 Manual labeling

Two software architects (including the author of this thesis) with more than five years of experience individually analyzed the extracted 2,259 issues in two steps. In the first step, issues were classified into two classes, namely “**Design Decision**” and “**Not A Design Decision**”. In the second step, the decisions identified in the first phase were manually classified into three decision classes, namely “**Structural decision**”, “**Behavioral decision**”, and “**Ban decision**” (cf. Section 5.3). These steps were not necessarily carried out sequentially, but as per the convenience of the experts. Before starting the labeling process, to ensure a common understanding between the two architects, a set of rules as shown in Table 6.5 were setup for manual classification. The classification of design decisions is purely based on the definition of decision categories as discussed in Section 2. To the best of author’s knowledge, there does not exist any design decisions dataset that can be used for reference. Hence, the rules shown in Table 6.5 was put forth to support the two architects for manually labeling design decisions.

Based on the aforementioned rules, both the architects manually analyzed the text in the summary and description attributes of all the extracted issues. Those issues with a missing description and whose intent was not explanatory using the textual description were marked as deleted. The issues that belonged to a specific decision category were labeled respectively, as well as, marked as a *Design Decision*. However, the issues that did not belong to any of the decision categories were marked as *Not A Design Decision*. During this process, we observed that some of the issues were abstract, in the sense that, they were broad issues that could be classified into more than one category. For example, the issue titled “Implement columnar in-memory representation<sup>5</sup>” aims to improve the memory efficiency of the system and represents a design decision. This issue affects the behavior of the system by introducing a new functionality and affects the structural aspects by introducing new Java classes for its implementation. In this study, multi-label classification was not applied and the focus was only on multi-class classifica-

<sup>5</sup><https://issues.apache.org/jira/browse/SPARK-12785>

**Structural decision:**

- + Adding or updating plugins, libraries, or third-party systems
- + Adding or updating classes, modules, or files (e.g., a class refers to a Java class)
- + Changing access specifier of a class
- + Merging or splitting classes or modules
- + Moving parts of the code or the entire files from one location to another (code refactoring to address maintainability issues)
- + Updating names of classes, methods, or modules

**Behavioral decision:**

- + Adding or updating functionality (methods/functions) and process flows
- + Providing configuration options for managing the behavior of the system
- + Adding or updating application programming interfaces (APIs)
- + Adding or updating dependencies between methods
- + Deprecating or disabling specific functionality
- + Changing the access specifiers of methods

**Ban decision:**

- + Removing existing plugins, libraries, or third-party systems
- + Discarding classes, modules, code snippets, or files
- + Deleting methods, APIs, process flows, or dependencies between methods
- + Removing deprecated methods

**Design decision:**

- + An issue that belongs to any one of the above categories

**Not a design decision:**

- + An issue that does not belong to any of the above categories

Table 6.5: Rules for manual classification of design decisions into structural, behavioral, and non-existence/ban decision categories

tion<sup>6</sup> and hence, the labeling of issues was restricted to only one label. Moreover, the majority of issues could be classified into one category since issues are typically concise so that developers can easily understand and implement the tasks. To sum up, architects were requested to mark issues belonging to more than one category as deleted since we argued that applying multi-class classification for detection and classification of design decisions is sufficient to validate the application of ML-technique to extract and classify design decisions.

Once the architects labeled all the issues individually, in a shared meeting, the training dataset was consolidated with two focus points:

- Issues that were marked as deleted by both the architects were removed from ADeX.
- All those issues that had inconsistent decision categories were also removed. Since inconsistent dataset results in unreliable classification results, this step ensured that the issues in the dataset were labeled correctly.

The labeling process resulted in a dataset with 2,139 issues with **781** issues labeled as Design Decisions and 1,358 issues labeled as Not A Design Decision. To avoid skewed results towards Not A Design Decision label (due to a higher number of issues labeled as Not a Design Decision), **790** issues labeled as Not A Design Decision were randomly selected for generating the design decision detection model. Furthermore, out of 781 design decisions, 226 were labeled as Structural, 389 were labeled as Behavioral, and the remaining 166 as Ban design decision. To ensure a balanced input for generating the ML model for design decision classification, **160** issues from each category were randomly selected.

#### 6.1.2.4 Setting up the machine learning pipeline

The pipeline shown in Figure 6.1 was used to generate the ML models for decision detection and decision classification. The pipeline itself was divided into two parts. In the first part – “process documents”, the labeled dataset was the input and the pipeline generated the term frequency representation of issues. The output of the first part was then consumed by the second part – “Generate model”, to produce the classification model and the result of applying the model on the testing dataset. Each issue in the labeled dataset was first tokenized to retrieve words. All the words were then transformed to lower cases. Stop words such as articles, conjunctions, and prepositions were removed. The remaining words were then stemmed to their root words using the Porter stemming algorithm [122]. Subsequently, a list of generated n-grams was appended to the word list. Generating n-grams helps to maintain the context of the usage of specific terms by considering its surrounding terms. For the evaluation, different values of n (from 1 to 5) was experimented and the corresponding results are documented in the next section. Finally, the list of words was converted into a term frequency representation. For decision detection with a labeled dataset of 1,571 issues (781 and 790 issues labeled as design decision and not a design decision respectively), the term frequency-inverse document frequency (tf-idf) was used for vector representation. The tf-idf representation evaluates the number of times a word appears in an issue but is offset by the frequency of the word in the corpus. However, for decision classification,

<sup>6</sup>Given that there are multiple labels, in multi-class classification, a document can be assigned to one and only one label. Whereas, in multi-label classification, a document can be assigned to any number of labels.

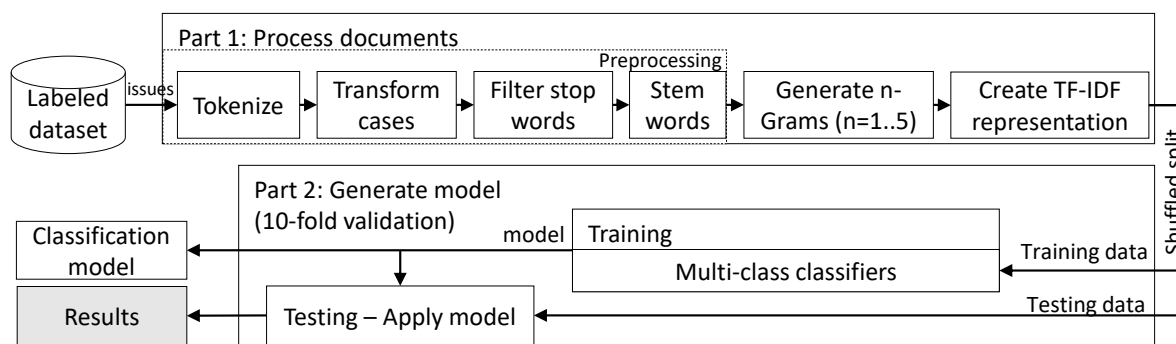


Figure 6.1: The machine learning pipeline for design decision detection and classification;  
 Classifiers: SVM, Naive Bayes, Decision tree, Logistic regression, One-vs-rest; n-grams: one to five; Split strategies: 90%, 80%, 70%, 60%, 50%;

only the term frequency (tf) was used as the dataset was comparatively smaller with 480 design decisions (160 issues in each decision category).

The term frequency representation of issues is provided as input to the second part of the pipeline for generating the classification model. Different shuffled split strategies (90%, 80%, 70%, 60%, and 50%) were used for observing the results. That is, documents were split into training dataset and testing dataset with different split percentages during multiple runs. Furthermore, the k-fold cross-validation technique was used in the model generation process for estimating the accuracy. In the test runs, 10-fold cross-validation ( $k=10$ ) was used. Using 10-fold cross-validation is common in data mining and machine-learning as it produces less biased accuracy estimations for datasets with small sample sizes [123]. Different multi-class classifiers were tested on the dataset with the parameters shown in Table 6.6. The classification model was then applied on the testing dataset to generate the classification results.

We implemented the pipeline shown in Figure 6.1 using Spark’s scalable machine learning library (MLlib) [125]. The MLlib component provides interfaces to create and execute the pipe and filter based pipelines. The pipeline with its configurations and the generated model was eventually persisted as a Spark model instance in ADeX for subsequent decision classification tasks. That is, for automatic detection and classification of newly created issues, this Spark model instance is executed and the classification label is persisted in ADeX.

The end-to-end workflow of the automatic design decision detection and classification was discussed in Section 5.3 (cf. Figure 5.17). Since the output of the first phase (decision detection) is the input to the second phase (decision classification), high accuracy of the results from the

<b>Support vector machines</b> – Kernel: linear; SVM type: C-SVC; Library: LibSVM [124] <b>Decision tree</b> – Criterion: gain ratio; Depth: 20; Confidence: .25; Minimal gain: .1 <b>Logistic regression</b> – Kernel: dot; ElasticNet: .8; Regularization: .001; Iterations:10 <b>One-vs-rest</b> – Base classifier: Logistic regression <b>Naive Bayes</b> – Additive smoothing: 1
---

Table 6.6: The configuration parameters used for training the machine learning classifiers

first phase is critical. The decision detection component loads the issues, uses the ML model generated for decision detection, and classifies each issue as either a decision or not a decision class. Next, the classification component takes the identified design decisions and classifies them into different categories using the decision classification ML model.

### 6.1.2.5 Evaluation of the ML pipelines for detecting and classifying design decisions

By applying the ML-based approach, we wanted to evaluate the following two hypothesis:

1. Design decisions can be automatically identified and extracted from issues.
2. Design decisions can be automatically classified into ADD categories, namely structural, behavioral, and ban decisions.

This subsection describes the validation or the invalidation of the aforementioned hypothesis. We present the results of applying different classifiers under different configurations for both decision detection and classification using the labeled dataset. In the given scenario of detecting and classifying decisions, the precision (fraction of automatically retrieved documents that are relevant) is as important as the recall (the fraction of relevant documents that were successfully retrieved). For instance, in case of decision detection, it is necessary that all issues that reflect design decisions are retrieved (high recall) and those issues which are not design decisions should not be automatically labeled as design decisions (high precision). Hence, the accuracy as the F-score [120], which is the harmonic mean of precision and recall is measured.

The multi-class classifiers namely SVM, Naive Bayes, Decision tree, Logistic regression, and One-vs-rest were evaluated. Since the logistic regression functionality provided by the Spark APIs could not handle polynomial labels, it was not used for decision classification but only for decision detection (binary)<sup>7</sup>. Split strategies from 90% to 50% and n-grams from one to five were analyzed. First, one could expect that varying the n-grams from one to five would increase the accuracy proportionally. That is, the use of patterns of words, which preserves the context of those words, should positively influence the accuracy of classification. Second, decreasing the split percentage from 90% to 50%, should reduce the accuracy substantially since lesser number of documents would be used for training the classifiers. In total, 25 individual runs (5 split strategies and 5 n-grams) were executed for each classifier and the corresponding precision, recall, and F-score were calculated. Finally, the average accuracy (average F-score) based on the arithmetic mean of the 25 individual runs for each of the classifiers was analyzed.

### Results related to the automatic detection of design decisions

	True decision	True not a decision	Class precision
Decision	<b>212</b>	18	92.17%
Not a decision	22	<b>219</b>	90.87%
Class recall	90.60%	92.41%	

Table 6.7: Decision detection: confusion matrix of the SVM classifier with a linear kernel

<sup>7</sup>cf. <https://spark.apache.org/docs/2.2.0/ml-classification-regression.html#logistic-regression>

The SVM classifier (average accuracy: 91.29%) outperformed Logistic regression (83.43%), One-vs-rest (79.45%), Decision tree (79.18%), and Naive Bayes (76.04%) classifiers. Since, the tf-idf representation of issues has a high dimensional feature space, sparse vectors, and few irrelevant features due to the data curation process, the SVM outperformed the rest of the classifiers. The maximum accuracy of 94.91% for the SVM classifier was achieved for a larger training set (90% split) with 3, 4, and 5 grams representation and the minimum accuracy of 87.4% with a smaller training set (50% split) and 1-gram settings. The confusion matrix for one specific execution run with 70% split and 3-gram configuration is shown in Table 6.7. This matrix depicts true and false positives as well as true and false negatives. The true positives (correct classifications) are highlighted on the diagonal of the confusion matrix. The precision for classifying an issue as a design decision is 92.17% and the recall is 90.60%. In addition, the precision for labeling an issue as Not A Design Decision is 90.87% and its recall is 92.41%.

Also, as shown in Figure 6.2, reducing the size of the training dataset (from 90% to 50%) the F-score decreases as expected but does not diverge more than 4% points from the average F-score of 91.29%. This indicates that the labeled dataset with 1,571 issues is sufficiently large enough to achieve a consistent F-score. Furthermore, it can be observed that the variation of  $n$  in  $n$ -gram generation does not drastically affect the F-score. As expected, the F-score is comparatively lower when the combination of words ( $n=1$ ) is not considered but the F-score slightly improves in the case of 2-grams and 3-grams. However, there does not seem to be any noticeable variations when  $n$  is greater than three.

To sum, *by using the linear SVM classifier along with  $n$ -gram ( $n \geq 2$ ) representation of words, design decisions can be automatically extracted from issues with an accuracy of 91.29% (hypothesis 1)*. Since no similar study exists with benchmarking results, 91.29% accuracy for automatic design decision detection can be considered encouraging.

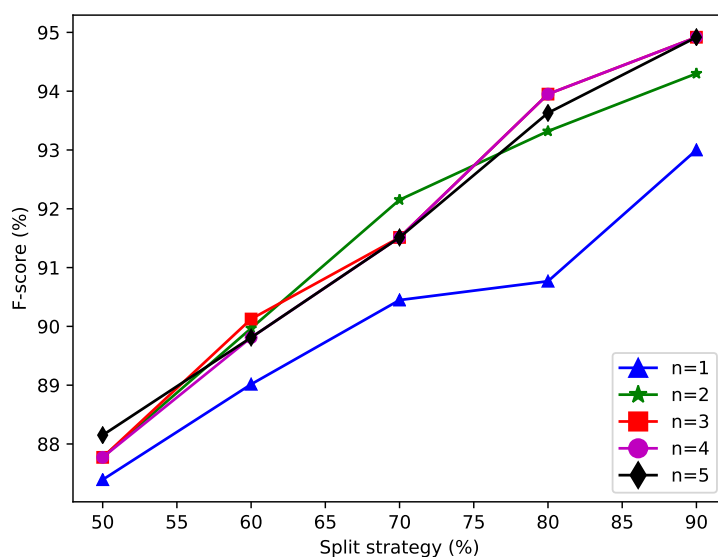


Figure 6.2: The influence of  $n$ -grams and split strategy on decision detection: increasing the training dataset increases the F-score and the variation of  $n$  in  $n$ -grams does not drastically affect the F-score



	True ban	True structural	True behavioral	Class precision
Ban	<b>45</b>	3	0	93.75%
Structural	4	<b>41</b>	13	70.69%
Behavioral	0	6	<b>39</b>	86.67%
Class recall	91.84%	82%	75%	

Table 6.8: Decision classification: confusion matrix of the SVM classifier with a linear kernel

### Results related to the automatic classification of design decisions

Even for automatic design decision classification, the linear SVM (average accuracy: 82.79%) performed better as compared to classifiers including Naive Bayes (59.09%), Decision tree (60.33%), and One-vs-rest (30%) classifiers. The confusion matrix for linear SVM with 70% training dataset and 30% testing dataset with trigrams is shown in Table 6.8.

Identifying ban decisions is critical, since they are typically not present in software artifacts. As shown in Table 6.8, the precision (93.75%) and recall (91.80%) for automatically classifying design decisions into ban decisions category are above 90%. On the other hand, the precision for structural and behavioral decisions are 70.69% and 86.67% and their recall values are 82% and 75% respectively. The lower precision and recall for structural and behavioral decisions is due to the existence of similar features (due to the classification rules presented in Table 6.5) in their corresponding training dataset.

As shown in Figure 6.3, reducing the size of the training dataset (90% to 50%) decreases the F-score as expected (89.9% to 76.2%). This variation is justified since the labeled dataset for decision categories is significantly smaller (160 design decisions in each category). On the

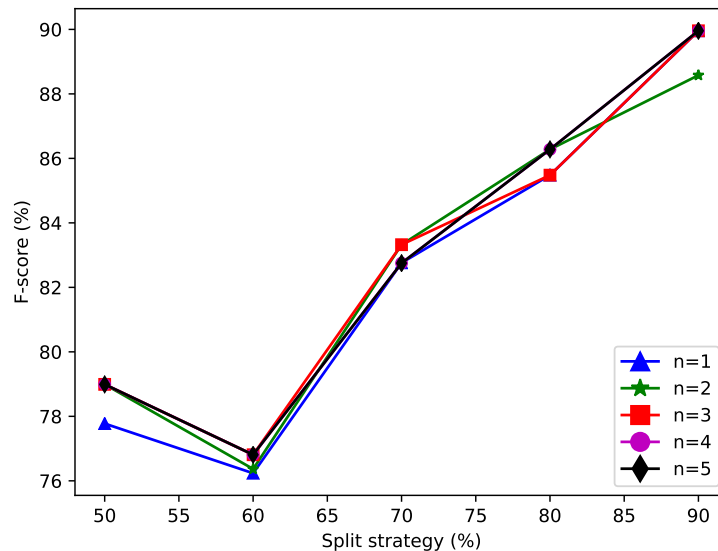


Figure 6.3: The influence of n-grams and split strategy on decision classification: increasing the training dataset increases the F-score and the variation of n-grams does not have any noticeable effect on the F-score

contrary, the variation of n-grams does not have any notable effect on the F-score. This indicates that the individual words within issues (or bag of words in text) play a significant role in the classification as compared to the usage of specific patterns of words and the context of the words. To conclude, *with the linear SVM classifier design decisions can be classified into structural, behavioral, and ban decision categories with an accuracy of 82.79% (hypothesis 2).*

### 6.1.2.6 Threats to validity

The results presented in the previous subsection are based on 1,571 labeled issues for design decision detection and 480 labeled design decisions for classification. The labeled dataset for classification is not as comprehensive as the dataset used for decision detection. Even though, we speculate that the generalization capabilities of design decision classification can be further improved by increasing the sample size of the dataset, providing relevant quantitative evidence is beyond the scope of this study. However, it should be noted that typically in ML-based approaches for text classification, increasing the sample size of the dataset substantially improves the classification performance [126].

The 1,571 labeled issues are extracted from two large OSS projects, wherein contributors have systematically maintained issues for more than three years. The hypothesis validated using the dataset might not be generalizable for projects where issues are reported scarcely. Hence, understanding what characteristics of the projects could influence the precision and recall of our approach are considered as part of our future work.

In the previous subsections, we have presented the results of automatic decision detection and classification independently of each other. However, if we consider the workflow shown in Figure 5.17, the accuracy of the decision detection affects the subsequent decision classification phase. In this study, we did not compute the accuracy of the end-to-end workflow.

Finally, as explained in the data curation process, we did not consider issues belonging to more than one ADD category. Considering such issues would require investigation of appropriate classification algorithms for multi-label classification and the study of the corresponding results.

### 6.1.3 Quantitative evaluation of the expert recommender

This subsection presents the evaluation of the expert recommender component that suggests appropriate architects and developers to address specific design concerns. The approach of using the expertise matrix was presented in Section 5.5.3. For the evaluation, we have used datasets of four different software projects that maintain issues in Jira. The two open-source projects (Apache Spark and Apache Hadoop Common) that were used in Section 6.1.2 for evaluating the ML-based approach have been used. And, the other two projects are closed-source projects from the industry partner. As shown in Table 6.9, the open-source projects have a higher number of unique contributors: Apache Spark has 95 unique contributors who made 447 design decisions and Apache Hadoop Common has 111 unique contributors who made 238 design decisions. The industrial projects are comparatively smaller, wherein, only 13 and 14 unique contributors have made design decisions in Project I and II respectively.

For analyzing the results of the individual datasets, the following strategy was applied:

1. Order design decisions based on the *resolution date*.
2. Split design decisions dataset into training [90% to 30%] and testing dataset [10% to 70%].
3. Use the training dataset to create the matrix.
4. For each design decision in the testing dataset, identify experts by matching the concept vectors against the matrix.
5. Open-source projects: Measure the precision at 5, 10, 15, 20, 25, 30, and max. The precision at 5 (P@5) indicates if an individual who actually resolved the design decision belongs to the top 5 results in the list of recommended experts. Here, *max* value refers to the total number of experts who can be recommended, that is, those individuals who resolved design decisions related to architectural elements *at least* once. Note that the recommended list is the list of top-n experts in the context of P@n where n is the list size.
6. Industry projects: Measure the precision at 2, 4, and 6. Since there are only 13 and 14 contributors in the industry projects, using a larger list will result in higher accuracy but will not lead to any interesting observations.

The overall accuracy of the expert recommendation algorithm (cf. Algorithm 1) can be calculated as the average of P@max across the investigated projects. As discussed in the subsequent subsections, even though the overall accuracy with P@max is higher than 60%, the average P@max varies depending on the project and hence, it does not provide valuable insights. However, understanding the behavior with smaller recommendations (P@5 and P@10) and observing the trend across different list sizes (P@5 to P@max) and split strategies is interesting for researchers to reflect on the influence of project characteristics on the recommendation system.

Furthermore, it should be noted that since design concerns were originally assigned to contributors without the aid of any system, the precision values should be interpreted as lower-bound estimates of the accuracy. Checking if an individual who actually resolved a design concern lies in the recommended expert list, indicates that new design concerns are assigned to individuals who have dealt with similar cases in the past. As demonstrated in the subsequent subsections, this assumption gets stronger as the size of the recommendation list increases.

ID	Name	Domain	Type	# design decisions	# unique contributors
1	Apache Spark	Data processing	Open source	447	95
2	Apache Hadoop Common	Distributed computing	Open source	238	111
3	Industry Project I	Connected Mobility	Closed source	368	13
4	Industry Project II	Knowledge management	Closed source	143	14

Table 6.9: The details about the dataset of four projects used for the evaluation of the expert recommendation system

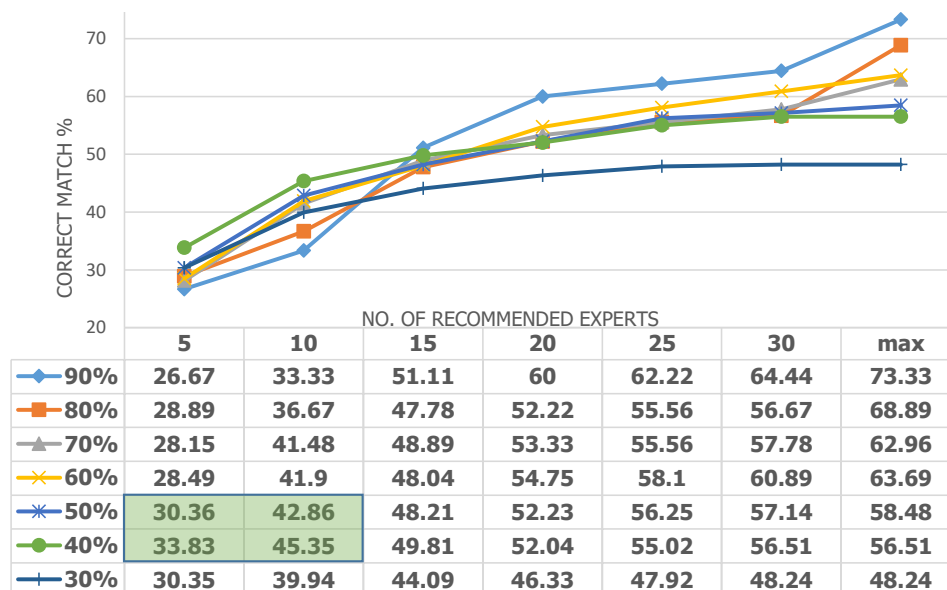


Figure 6.4: The evaluation results of the expert recommender for the Apache Spark dataset

### 6.1.3.1 Evaluation of the expert recommender using the Apache Spark dataset

Among the datasets, the Apache Spark dataset has the largest number of labeled design decisions (447) with 95 unique contributors who resolved those design decisions. As shown in Figure 6.4, increasing the size of the training dataset for creating the expertise matrix (from 60% to 90%) and the size of the recommendation list (see from P@15 to P@max), the precision also increases. This is rather intuitive as one could imagine that increasing the size of the training dataset as well as the solution space (expert list), the accuracy must also increase. However, corresponding to the results – P@5 and P@10, using larger training datasets **decreases** the accuracy. The reason for this is that when a larger training dataset is used, the values of expertise atoms gets distributed across the corresponding expertise matrix and the individual who resolved a design decision in the testing dataset might not be present in the recommended list of more qualified personnel. On the contrary, when a smaller training dataset (40% and 50%) is used, the expertise matrix is rather concise and results in higher accuracy (for P@5 and P@10) as compared to a larger training dataset. This is an important observation, since, for a 100 members team of architects and developers, recommending more than 5 to 10 key experts who should be involved in the decision-making process might be an overhead. Hence, it is necessary to consider an optimal size of the training dataset to prevent an overfitting of the expertise matrix and to subsequently use it for recommendations. In case of the Apache Spark project, using 40% or 50% of the dataset (approx. 200 design decisions) is sufficient to recommend experts for ADM.

Furthermore, reduced P@5 and P@10 values with a larger training dataset indicate that these design decisions are not made by a selected few individuals but is well distributed among the architects and developers. As it should be in an ideal case, this indicates a “healthy” project where knowledge does not reside only with a few experts.

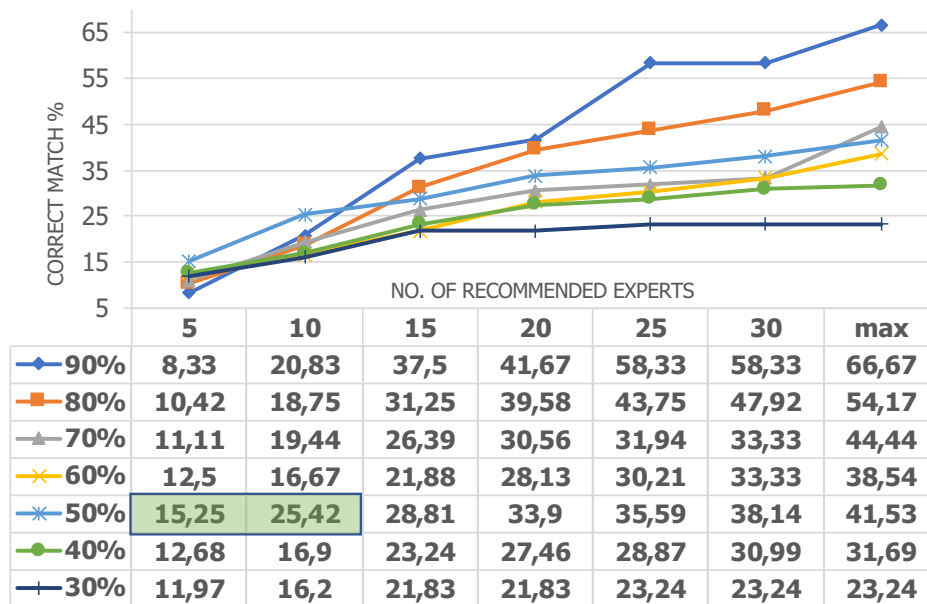


Figure 6.5: The evaluation results of the expert recommender for the Hadoop Common dataset

### 6.1.3.2 Evaluation of the expert recommender using the Hadoop Common dataset

The Apache Hadoop Common dataset comprises of 238 design decisions which have been resolved by 111 unique contributors. As shown in Figure 6.5, the results for this dataset are similar to the results of the Apache Spark dataset.

- Increasing the size of the training dataset (from 60% to 90%) and the recommendation list (from P@15 to P@max) increases the accuracy.
- For P@5 and P@10, the smaller training datasets (50%) outperforms the larger dataset.
- Lower P@5 and P@10 values for larger training datasets indicate a “healthy” project where design decisions are not made by a few architects and developers.

Similar observations from both these open-source projects indicate that even a smaller design decisions dataset is sufficient to build a comprehensive expertise matrix to recommend experts who could be involved in the DDM process.

### 6.1.3.3 Evaluation of the expert recommender using the Industry Project I dataset

Unlike the open-source projects wherein stakeholders have maintained issues in Jira since 2012, the industrial projects are under development and maintenance since 2016 and the team size of architects and developers is considerably smaller. The first project under consideration is the connected mobility lab (CML) project. This project aims to provide mobility-related services for commuters in metropolitan areas by benefiting from the sensor data collected from different means of transportation. Stakeholders of this project have captured 1,233 issues in Jira. Using the decision detection model, 368 design decisions were identified which were resolved by 13

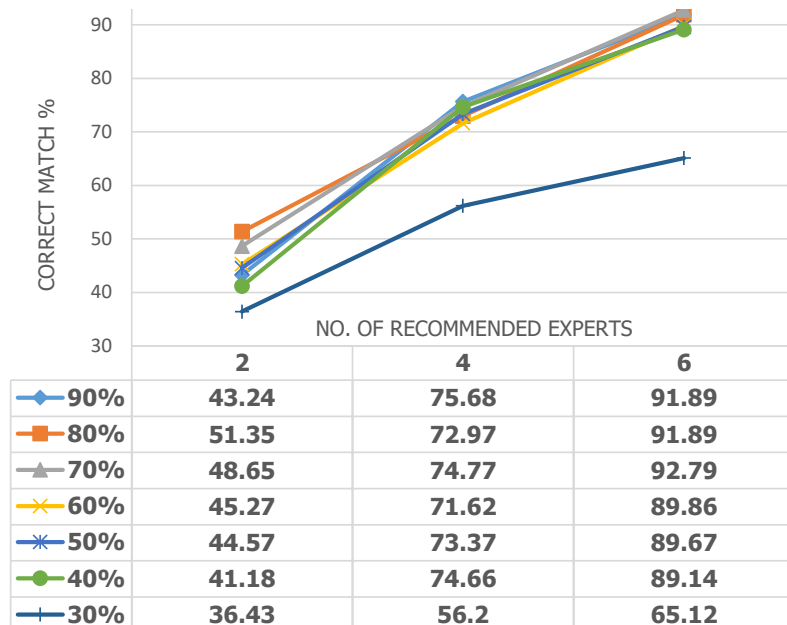


Figure 6.6: The evaluation results of the expert recommender for the Industry Project I dataset

unique contributors. After creating the expertise matrix using the training dataset and matching the concept vectors of design decisions in the testing dataset, the precision at 2, 4, and 6 was measured. Note that,  $n$  in  $P@n$  was restricted to 6 due to fewer contributors. Since this project has a large number of design decisions (368), as shown in Figure 6.6, the precision values do not vary significantly for different split strategies. Similar to the open-source projects, increasing the size of the training dataset and the size of the recommendation list also increases the accuracy of the recommendation system.

Contrary to the open-source projects, higher accuracy for  $P@2$  and  $P@4$  was observed in case of industrial projects (cf. Figure 6.6 and 6.7). The average accuracy ( $P@2$  and  $P@4$ ) across different splits is 44.38% and 71.32% respectively. That is, in 44% of the cases, either of the two individuals who had the most expertise actually resolved the design decision. Similarly, 71% of decisions were resolved by the top four individuals with most expertise. Surprisingly, either one of the top two recommended experts remained consistently in all the recommendations.

Even though the higher accuracy indicates that the system can identify experts who can deal with specific decisions, the fact that only those individuals actually resolved most of the decisions is not “healthy” for the project. It shows that there are only a few individuals with relevant expertise and the chances of knowledge vaporization in case they leave the project is higher.

#### 6.1.3.4 Evaluation of the expert recommender using the Industry Project II dataset

The second industry project that was analyzed is a knowledge management system which guides stakeholders during different phases of the application lifecycle of software projects. Stakeholders of this project have maintained 1,153 issues in Jira since early 2016 and the decision detection

model identified 143 design decisions which were resolved by 14 unique contributors. As shown in Figure 6.7, the results are similar to that of the first industry project:

- The average accuracy of finding experts in the top 2 and top 4 recommendation list is as high as 58.60% and 73.77% respectively.
- Higher accuracy (P@2 and P@4) and consistently recommending either one of the top two experts indicate that decisions were made only by a few individuals and there is a need for knowledge transfer within the team.

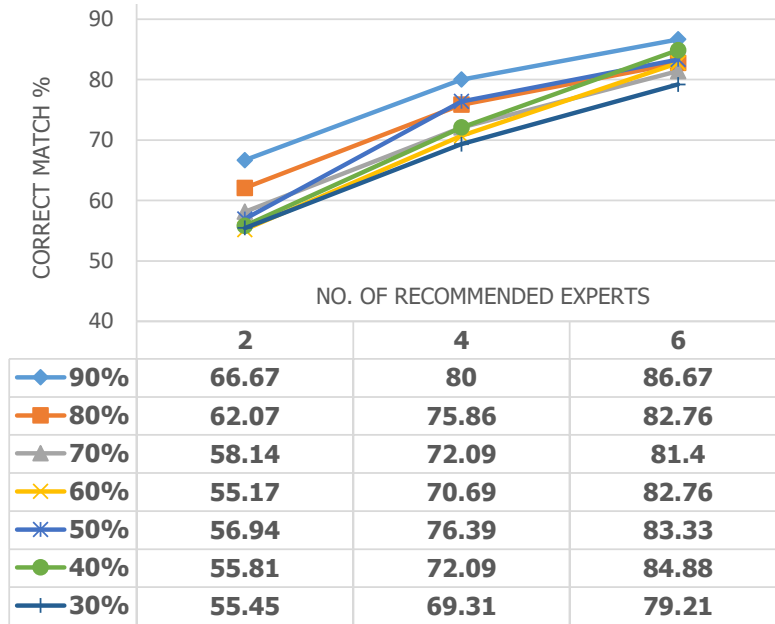


Figure 6.7: The evaluation results of the expert recommender for the Industry Project II dataset

### 6.1.3.5 General observations and threats to validity

#### A. Project and team characteristics

As presented in previous subsections, we observed higher accuracy with smaller recommendation lists for the industry projects as compared to the OSS projects. This is due to a couple of factors, namely, the *number of contributors* and the *culture of assignment of issues*. Typically, teams of small and medium-sized projects (with 10 to 50 architects and developers) contain only a few architects and senior developers. The key design decisions including setting up the IT infrastructure, selecting the communication protocol, and making changes to the data models are made by those experienced architects and developers. Hence, for new design decisions, the proposed algorithm correctly identifies those experts in such smaller teams as compared to larger teams where design decisions are resolved by many contributors. Furthermore, in the OSS projects, contributors independently resolve issues (bottom-up) by submitting a pull request in the code repository which is then merged into the main branch and the respective issue is closed. Whereas, in the industry projects we analyzed, issues are assigned to developers by architects

or senior developers (top-down). Hence, contrary to the industry projects, in the OSS projects where developers had more freedom to choose the design problems, we observed that the values of the expertise atoms were scattered across the expertise matrix. To avoid the risk of knowledge vaporization, we prescribe the use of the expertise matrix to identify hotspots within the matrix (rows containing darker cells) early in the project so as to ensure the involvement of junior developers while addressing design concerns.

Within the scope of this study, we have not addressed the *cold-start problem* in the recommendation system. That is, when new contributors join the team, currently, it is not possible to automatically create their expertise profiles. However, we consider this to be a technical challenge as one could integrate external data sources such as professional resumes and (public and organizational) social networks to extract their skills.

Another shortcoming of the matrix-based approach is that we have to consider an optimal size of the training dataset for creating the expertise matrix. As discussed in the previous subsection, using a larger training dataset might not result in better accuracy. The size of the training dataset has to be dealt on a project-to-project basis. Within the scope of this study, *we could not generalize the optimal size of the training dataset*.

One of the frequent concerns raised by our industry partner is “how to deal with issues captured in different languages”. In some projects, either all the issues are in the German language or there is a mixture of both the English and the German text. *Dealing with such scenarios is challenging* as we not only have to use a translation service but also retrain the decision detection models which is time and effort intensive.

Finally, during the demonstration of the results to the stakeholders of Industry Project II, one of the architects expressed that it would be beneficial to apply the approach across the organization’s projects so as to know with whom one has discuss for resolving similar design problems. Addressing this point is not a technical problem but it is difficult since *every project within an organization has different confidentiality criteria*. Moreover, getting the approvals to conduct such studies and to demonstrate the benefits of such approaches to the respective stakeholders is both a time-consuming and a “political” activity within large organizations.

### ***B. Expert recommendation***

The architects from Industry Project I highlighted that the system should also consider attributes including availability and workload of experts as well as criticality and priority of design decisions. Since projects maintain such structured information in Jira, *we should consider these complimentary parameters while generating the list of experts*.

Furthermore, it is necessary to emphasize that the aim is not to automatically assign contributors to address a design concern but to recommend a list of experts who could be involved in the ADM process. In this context, first, we need to have a balanced mix of both senior and junior architects and developers in the list so as to ensure knowledge transfer. Second, it is not sufficient to present only the list of experts but we also need to *identify and assign roles to the experts* such as owners, decision-makers, and moderators. We are currently investigating with our industry partner the aforementioned aspects of improving the quality of the recommendations as part of an of a follow-up research project under an AI4AM initiative.



### C. Personal experience and cognitive biases

In the expertise matrix based approach, we made the assumption that architects and developers - intentionally or unintentionally - rely on their experiences while making design decisions or even when selecting a design problem to be addressed. The use of experience as an “anchor” while making decisions may lead to anchor and confirmation cognitive biases [127]. The qualitative interview-based studies [128] and [39] have indicated personal experience to be a key factor influencing the ADM process. The recommendation results for the OSS projects show that there is a high chance (cf. P@max; avg. 61.72% for Spark and 42.89% for Hadoop) that contributors select similar concerns that they have addressed in the past. Similarly, in the industry projects, issues reflecting design decisions are assigned to those developers who have dealt with similar concerns in the past (cf. P@6; avg. 91.22% for Industry Project I and 82.45% for Project II).

The aforementioned observation provides *quantitative evidence to indicate that experience of architects and developers play an important role when they select and resolve design concerns* which in turn influences the ADM process.

## 6.2 Qualitative evaluation of ADeX in real-world projects

This section presents the feedback received from project partners in the industry during internal presentations. The discussion in this section consolidates the feedback from 15 software architects and developers received during open discussions in five different presentations. During these presentations, ADeX was demonstrated to the audience. Due to time constraints, the project data was preprocessed prior to the presentations. For instance, during the demonstration of the results for Industry Project II discussed in the quantitative evaluation section (cf. Section 6.1.3.3), the issues from the project had already been imported into ADeX and the pipeline discussed in Section 5 had already been executed. Feedback received during and after the presentations are grouped into the following categories:

### Meta-model based AKM framework:

Contrary to this author’s intuitive opinion that stakeholders appreciated/understood the benefits of a **meta-model based system** for architectural knowledge management, it was rather difficult to convince the need for such a system. During one of the presentations at a conference<sup>8</sup>, one of the audiences raised the following question “If the focus is on supporting software architects during the decision-making process by extracting design decisions from Jira, why do you (referring to this author) need a meta-model based system/approach?”.

Further convincing was required to emphasize the the fact that each project maintains it’s artifacts using different concepts and to enable quick adaptation of the software system, the domain models (in this particular scenario - dynamic knowledge model) need to be configurable at runtime. This follows the idea of models at runtime. For instance, if stakeholders of one project use the term Issue, in another project they might use the term Iask. In such cases, maintainers of the software system (ADeX) should quickly be able to change the domain model. These changes not only relate to the name of the concept in the domain model, but also includes

<sup>8</sup>15 minutes Q&A session during the doctoral symposium presentation of the paper [61] at the ECSA 2017

relationships with other concepts, their usage in rules (model-based expressions), as well as their corresponding user interfaces. The meta-model based framework presented in Section 5.1 facilitates such adaptations at runtime. Contradictory to model-driven approaches (for example, using eclipse modeling framework), model-based approach does not require recompiling the source code to make those adaptation; both the domain model and its corresponding data can be updated at runtime using a web interface (cf. [6]).

### **SyncPipes - the data synchronization platform:**

The idea of reusable and configurable adapters is not new. However, since in 2015, there did not exist a platform for developing and reusing adapters to extract data from *REST-based services*, transform the data, and then load the data into target systems, the need for SyncPipes to integrate and synchronize data into ADeX was well received by the industry partner. Once the adapters for importing data from Jira, GitHub issue tracker, and Enterprise Architect were demonstrated to the industry partner, a development team took over the further development of SyncPipes adapters. The adapters for Team Foundation Server to extract issues and MS project to import project tasks were independently implemented by the development team.

### **Decision detection and classification:**

Once a design decision is identified by the decision detection ML-model, that design decision is classified into one of the three categories, namely, structural, behavioral, and non-existence/ban decision. These decision categories are based on the taxonomy defined by [18]. These design decisions can also be classified according to their abstraction level. That is, as suggested in [16, 129], design decisions can be made at three level, namely, high-, medium-, and low-level design decisions. During one of the presentations, architects suggested that the classification based on the abstraction level would be more beneficial for them. That is, it will allow them to view and analyze only high-impact (high-level) design decisions. Classifying design decisions based on the abstraction level is a valuable feedback and is considered as part of future research. One approach to address this problem would be to follow the similar approach for decision classification presented in this dissertation. Instead of labeling design decisions based on the categories suggested by [18], they would have to be labeled according to their abstraction levels and the models would have to be retrained thereafter.

The second feedback or the area of improvement concerning the classification of design decision relates to the accuracy of the classification model. As presented in the evaluation section 6.1.2, the accuracy specifically for structural and behavioral decisions is comparatively lower than ban decisions. The reason for lower accuracy being that, sometimes, it is difficult even for a human to distinctively classify a decision as structural or behavioral. A behavioral change to a system, more often than not, causes structural change and vice-verse. Hence, in principle, instead of labeling design decision specifically into one decision category (cf. discussion presented in Section 6.1.2), one should label decisions into multiple categories. That is, a multi-class classification model must be used. This is a shortcoming of the ML-learning based classification approach presented in this thesis, which not only the author is aware of, but was also highlighted during discussions with stakeholders. The argument being that manually labeling documents is a time and effort intensive task and within the time constraints and scope of this study, to prove

the feasibility of applying ML in the given context, only a multi-label classification was applied. The application of multi-class classification is subject to future research.

#### **User interface related to quality attributes:**

The bar chart showing the number of design decisions addressing specific quality attributes and its evolution was rather an eye-catching feature during the presentations. During the presentation, the results of rationale extractor component was both appreciated as well as critiqued.

On the right side of the UI (as presented in Figure 5.38), the list of quality attributes which were not tagged with any of the design decisions are listed. During demonstrating the results for the case study - Industry project II, one of the architects mentioned that “we are not missing those quality attributes, since they are not related to our project”. The term “missing” rather puts forth a notion of “blaming” an architect for not considering a specific quality attribute. Hence, there is a need for further investigation on how to represent those quality attributes that do not have any discussions related to them within design decisions.

During the same presentation, many of the segments within the bar chart were investigated by reasoning about their corresponding design decisions. It was observed that even though most of the design decisions were tagged with correct quality attributes, architects present in the discussion pointed out a few discrepancies. For instance, “decision X does not relate to performance but instead it is about addressing availability concerns”. The rationale extractor component is based on very basic keyword matching approach and hence showed false positives. As discussed in Section 2, there already exists a plethora of work in the direction of classification of non-functional requirements. Since, in this dissertation, not much focus was given specifically to the topic of rationale extractor, it is indeed the case that some of the design decisions are tagged with incorrect quality attributes. However, by building on previous research, we believe that the rationale extractor component can be further improved to meet stakeholders’ needs. The same was also communicated to the stakeholders of the industry project I.

#### **User interface related to architectural elements:**

There were two main suggestions for improving the user interface with the bubble chart of architectural elements (cf. Figure 5.39). In the current implementation, each bubble represents an architectural element and there does not exist any relationships between the nodes. It was suggested that highlighting the relationships between nodes using edges might be helpful for architects to understand the dependencies between different architectural elements. For instance, an architectural element such as Java can be linked with JDBC and JUnit.

The second suggestion, similar to the first one, also deals with how architectural elements are related to each other. Architectural elements belonging to similar categories (genres) can be grouped together. Doing so, will simplify the visualization with fewer number of architectural elements. If end-users are interested in a specific architectural element, then, they can drill-down by clicking on the high-level architectural element. For example, if there exists architectural elements such as application server, Apache Tomcat, and Glassfish server, then these three elements can be grouped together. Realization of such a categorization should not be effort intensive, as the required information (rdf:type - genre, concept, or work) is already available from the DBpedia ontology (cf. discussion in Section 5.5.1).

### Expert recommender system:

The architects and developers found the UI of the expertise matrix (EM), shown in Figure 5.40, a useful tool to quickly lookup (using the search functionality) individuals with expertise in specific architectural topics. During the presentation of the results for Industry project II, architects and developers argued if a specific individual was indeed an expert on a specific topic. We found the conversations “political”; in the sense that, for those individuals who were not present in the presentation room but were part of the project, it was argued that they did not contribute “much” to that architectural element and hence they should not be an expert. However, such comments were not made against each other who were present in the audience. In essence, software engineering generally is “people’s” problem. How one person considers whether another person is an expert on a specific topic depends not only on quantitative numbers but also on personal relationships. Hence, further investigation is needed, to better understand the social network of architects and developers and incorporate respective metrics within the EM.

During the discussions, an important concern was raised against the use of an expertise matrix to quantify and measure the the expertise of architects and developers. At least three architects commented the following: “some people [architects and developers] might **not be happy** that you are measuring their expertise”. The notion that such an expertise matrix could potentially be used for or against an individual by the upper management is indeed a concern. Special care must be taken before using such a system in industrial settings. Appropriate approvals from all involved parties regarding monitoring their activities (for instance, in Jira) and quantifying their expertise about architectural knowledge must be made upfront.

Concerning the view shown in Figure 5.41, which lists the experts who could be involved in addressing certain design concerns, three specific comments were made. In the current UI, along with the expert in the recommendation list, their corresponding expertise score is displayed. One of the architects, during the presentation, commented that “showing the expertise score leads to saying that person A is better or worse than person B; this is not nice.” This concern too is similar to the “people’s problem” discussed before. In general, ADeX needs to be further evaluated so as to ensure such minute details do not hinder its actual benefits.

A lead architect involved in Industry project I commented the following: “Our project scope has reduced over the years and we are less than ten team members. I am not sure, how useful this is for us. We all know whom to talk to when we want to ask something. Anyway, this helps us to improve the process of task assignment. I see that I am recommended as an expert in many case. I did not actually solve those issues. During our meetings, for documentation, I only added myself as an assignee and resolved those issues”. Regarding the first point, we agree that such a recommendation system makes more sense for large software projects with more than [at least] fifty software architects and developers. Concerning the second point, the same lead architect also added “it might be interesting to consider not only the assignee of an issue, but also those individuals who changed the status of that issue, commented within the issue, or who made the changes in the code”. Since, issue management systems such as Jira are very powerful in maintaining such rich information in a structured manner, it is technically possible to identify and consider these aspects. Investigating the changes at the code level depending on a specific issue rather depends on the project itself. In some projects, stakeholders maintain the pull request that resolved the issue or specify the issue ID in the Git commit. This information

can be considered while creating the expertise matrix. The investigation of the aforementioned scenarios is considered as part of future research.

Architects from the Industry project II suggested the following concrete improvements. Firstly, they suggested to have a predefined team of experts who could be assigned to resolve design concerns. Second, in the list of recommended experts, they suggested to include novices. Since, including novices might help reduce the chances of knowledge vaporization. Thirdly, since, architects' time is very precious and the number of architects within a project is usually limited, they also suggested to indicate the availability and work pressure of the recommend experts. Based on those factors, a project manager or a scrum master can pick suitable and available architects and developers to resolve design concerns. These suggestions were incorporated in a prototype developed as part of a Master's thesis [117].

### Recommendation of similar design decisions:

In a Master's thesis [118], we observed that it is possible to identify different types of similar design decisions. The recommended list of similar design decisions includes "related to", "sub-task", "same as", and "required by". Even before highlighting the aforementioned fact, the lead architect of Industry project I commented: "In Jira, we do not maintain relationship between issues. I feel these similar decisions can be used to enrich the issues with links. I think this is a more useful feature [compared to expert recommendation] for us".

Even though, the decision clustering component only identifies similar design decisions and does not indicate the type of relationship, finding the type of relationship would be beneficial not only for documentation but also to create decision graph for impact analysis.

The development team involved in the Industry project II is currently investigating the topic of identifying similar issues in issue management systems using document clustering approaches.

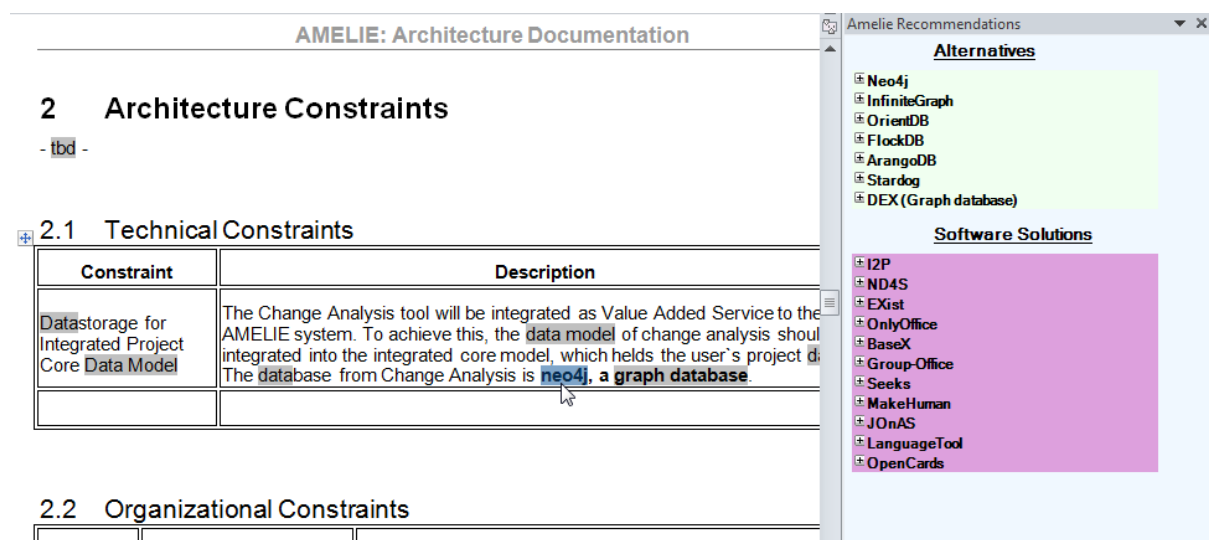


Figure 6.8: A Microsoft Word plugin for architectural recommendations: the plugin uses the recommendation services presented in this dissertation

### Annotation of architectural elements in textual description:

The team driving the AKM research within the industry partner suggested to use the services from the architecture annotator component in different ways. To better integrate the process of annotating architectural elements and for providing recommendations to architects in their day-to-day activities, the first suggestion was to embed the services into Microsoft Word. Since, architects use Word to document the architectural description in their projects, it would be beneficial if they can receive recommendations “on the fly” while writing those documents instead of uploading those documents to a web client. Hence, as shown in Figure 6.8, a Word plugin was implemented that used the services provided by the Akre-Server component. New extensions within the pipe-and-filter architecture of the existing UIMA pipeline were also added to annotate custom organization-specific terms as well as terms used in SECO pattern catalog [130]. The recommendations about experts were also presented to architects using this plugin.

The infrastructure provided by the UIMA framework allowed incorporating new requirements related to the annotation of documents (in this case, in the Atom editor). In the second iteration, to address the challenges faced by a testing team within our industry partner, the UIMA pipeline was extended with a regular expression (regex) annotator. The testing team in a business unit maintained more than 17,000 test cases and many of these test cases contained code smells. These code smells could be identified by a set of rules. For example, “a test case should not hardcode the name of the machine” [patterns of machine names were given]. Since test cases were written in a variant of Visual Basic (VB) language and the Atom editor supported VB, an Atom plugin was implemented to highlight the code smells within test cases. Using this plugin, it is also possible to add custom regex to annotate textual documents.

In the last and the ongoing iteration, a chat application is being developed which uses the already existing services provided by the Akre-Server component. As shown in Figure 6.9, this chat bot - NCK bot (named after the business unit within the industry partner who plan to use it in production) answers different questions raised by the stakeholders. Since, the stakeholders of

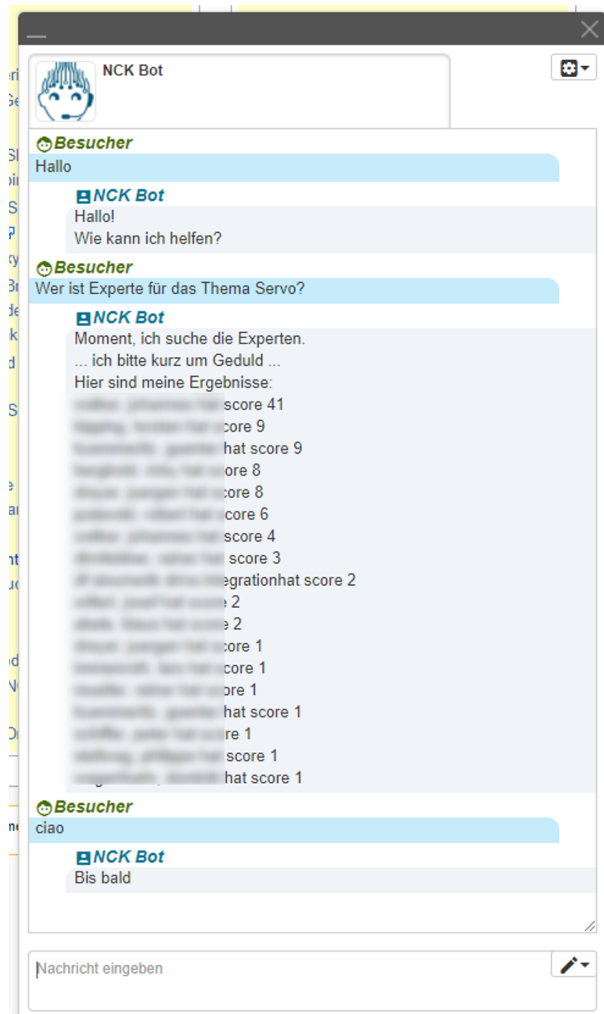


Figure 6.9: A chatbot for architectural recommendations

---

this project used German as the main language for communication, the Akre-Server component was modified to process German natural language text. Using this chat bot, end users can find answers for the following question:

- Find experts for a specific topic
- List issues concerning specific quality attributes
- Find issues that are similar to another issue

The main intent behind the use of a chat application is to integrate the recommendation services into the daily activities of architects and developers and to provide an easier means of communication with the recommendation system. Given that architects and developers frequently use chat applications to communicate with each other, they can use the same system to get help, as and when required, without any overhead of shifting to a different system and perspective.

### 6.3 Evaluation summary

Section 6.1, presented the quantitative evaluation of various approaches that form the core of different components within ADeX. The quantitative evaluation provided concrete values for the accuracy of the described approaches including, the ontology-based, ML-based, and expertise-matrix based recommendation approaches. On the other hand, Section 6.2, summarized the feedback comprising of areas of improvement, positive, as well as, negative aspects of ADeX as perceived by software architects and developers in industry. Along with the feedback, few lessons learned during the presentation were also discussed. Detailed discussion on the lessons learned is elaborated in the next chapter.

Concerning the qualitative evaluation, it can be argued that a systematic qualitative evaluation of the end-to-end system was not performed. However, it should be noted that, this research project in collaboration with the industry partner was started in 2015 and is an ongoing endeavor. The components and services within ADeX are conceptualized and realized iteratively based on the feedback received both from stakeholders in business units of the industry partner as well as from researchers in the software architecture community. Each component followed more than one iteration of Hevner's three-cycle design science method [131]. Hence, qualitative assessment is an inherent and integral part of the design and development phase of ADeX. It should also be noted that, apart from the feedback discussed in the previous section, a Master's thesis project [78] was also conducted to improve the usability of the system. In that thesis, usability was not restricted to the usability of the user interfaces but also covered aspects such as understandability and maintainability of ADeX for the developers and future maintainers. Even in that thesis, detailed requirements were collected to improve the user interfaces of the system. Once the user interfaces were adapted, they were again presented to the stakeholders in the industry to validate if the requirements were met.

In essence, since ADeX is an ongoing research project that facilitates new ideas for supporting software architects and developers in the architectural decision-making process, feedback (ground-truth) based on the application of the system in real-live projects will continue to play a major role in how the system evolves in the future.





Over the last four years, with frequent interactions with architects in our industry partner as well as researchers in software architecture research community, we have received many feedback and suggestions for the improvement of the ADeX system. This chapter presents those improvement areas and our future research roadmap. We distinguish between lessons learned from our experiences with our industry partner and lessons learned from interactions with researchers at ICSA and ECSA conferences from 2015 to 2018.

## 7.1 Lessons learned

### **Lessons learned from our industry partner**

*Lessons learned 1:* Initially, ADeX was developed as a standalone application which provided various services and user interfaces as presented in the previous chapters. We soon realized that architects instead prefer services to be integrated with those systems that they frequently use. For instance, plug-in in Microsoft Word to support architects while they are writing architectural documents, plug-in for IDEs while they are implementing a solution, or a chatbot integrated with wiki systems while they are maintaining architectural documents. Early on in the project itself, we considered this critical aspect and provided those services in the respective clients as discussed in Section 6.2. As part of our future work, we intend to conduct user experience studies to reflect on the effectiveness of such integration.

*Lessons learned 2:* This dissertation only focuses on the text analysis of the project artifacts to support architects during the decisions making process. Consideration and analysis of other data formats including audio from architectural meetings and images in architectural documents as well as whiteboards are valid inputs and concerns from our industry partner. In this direction,

we recently completed a Master thesis project titled “Automatic documentation of results during online architectural meetings<sup>1</sup>”. Given the fact that many of the high- and medium-level decisions are made in groups during architectural meetings, this project aims to investigate how a virtual meeting assistant can support architects during those meetings by analyzing the audio input during live architectural meetings.

*Lessons learned 3:* The quantitative evaluation of different recommendations presented in this dissertation was performed by considering datasets from both open-source and closed-source projects. However, the evaluation of the recommendations was performed only within the project and not across the projects. That is, for instance, the question of whether the ML model trained using the dataset of Apache Spark project can be used to detect decisions in Apache Hadoop project has not been investigated. In other words, the reuse of AK across projects (in different domains) remains to be investigated. At our research department, this topic of reuse of AK across projects will be taken forward by our newly joined research associate<sup>2</sup>.

*Lessons learned 4:* Establishing tractability across software artifacts has been an ever-green problem in software engineering, and our interactions with the business units at our industry partner have also suggested the same. One of the frequent questions often raised during the demonstration of the ADeX system is that “can we trace the identified design decisions to the actual implementation”. The fact that many of the taken decisions are never realized, duplicated (UC 43, cf. Section 3.2), or changed during the implementation, is not reflected in the documentation. These aspects introduce new challenges and future research directions in the context of this dissertation. An aspect that we observe as a low-effort investigative step, which we aim to investigate next, is the idea of leveraging from the conventions typically followed in well-maintained open-source projects (e.g., Apache projects). In many of the Apache foundation projects, we observe that developers either maintain an explicit link between an issue in issue management system to source code commits in versioning systems (or vice versa) or maintain such information in the comment descriptions. Extracting these links might also help to establish traces between design decisions (since they are identified from issues) to their actual implementation. Such a mechanism would then open new ways to explore ideas including the impact of design decisions (UC 39), effort estimation for realizing similar decisions, and checking the implementation against made design decisions (UC 40).

### **Lessons learned from the research community**

*Lessons learned 5:* In contrast to our observations with our industry partner, we notice that researchers, especially since 2012, have been more inclined towards the process of ADM rather than towards tooling capabilities. Our analysis of the literature review indicates that from 2005 to 2011 there was a fair amount of publications presenting AKM tools that supported documenting design decisions. However, since then, the focus seems to have shifted to better understanding how architects make design decisions. Many topics are recently being investigated in depth: GDM, factors such as cognitive biases influencing ADM, and the effect of uncertainties in decision making. Within the scope of this dissertation, we have only managed to share some of our insights on the topic of cognitive biases in ADM (cf. next section for a discussion) and the use of uncertainty expressions in GDM (cf. [70]).

---

<sup>1</sup><https://www.matt.hes.in.tum.de/pages/1e011s37ghgvz>

<sup>2</sup><https://www.matt.hes.in.tum.de/pages/v2o7t4t0vg87>

## 7.2 Ongoing research activities

In the final phases of this dissertation, two specific topics were explored. The first topic, related to ADM, was covered to ensure the completeness of the discussions on ADDs. Second, we investigated the possibility of extending our approach to support architects and developers by analyzing the real-time audio streams in online meetings. We briefly summarize these two activities in this section.

### Decision making process models and cognitive biases

Any discussion on ADDs cannot be complete without covering the process of making those decisions. The process of making ADDs is referred to as the ADM process and reflects on the “how” aspect of an ADD. To better understand different decision-making process models and to document the influence of human cognitive limitations on the steps within the process models, a master’s thesis [132] was carried out in our research department<sup>3</sup>.

**Note:** This section has been taken from our study documented in the publication titled “Decision making and cognitive biases in designing software architectures” [133].

In this study, we observed that two Decision Making Models (DMMs), namely, behavioral (NDM) and normative (RDM) models are frequently discussed in software architecture literature [40, 41, 71, 134, 135]. We also saw references to Bounded Rational Model (BRM) for ADM [39, 59, 136].

The decision-making process that follows a sound logical reasoning belongs to the normative decision-making approach. Kahneman refers to normative decision making as System 2 thinking which is slower, deliberative, and logical [71]. Normative approaches such as the Rational Economic Model (REM), the Brunswik’s Lens Model, and the Cynefin framework are typically applicable in *ideal* scenarios (“ideal decision-making process”) wherein requirements with no future changes and resources such as time, budget, and team dynamics are available a priori.

The Brunswik’s Lens Model [137] helps decision-makers to determine the optimal decision based on the statistical weights assigned to the factors and clues influencing alternative decisions. It requires an optimal decision and the corresponding weights to begin the process and to compare it against the actual decision. The Cynefin framework [138] is useful for executives and policy-makers to make sense of situations for decision making. It provides decision-makers guidelines on how to analyze situations in five different context, namely, simple, complicated, complex, chaotic, and disorder. The REM [139] comprises of a series of steps for decision-making. Decision-makers (a) define the concerns, (b) list the alternatives to address a concern, (c) rank and assign weights to the alternatives, and (d) choose the optimal alternative based on the weights.

It should be noted that while normative approaches work well in ideal situations, it is difficult for architects to use it since they work under various real-life constraints such as time, complexity, and permanently changing budget constraints. On the other hand behavioral approaches that reflect real-world scenarios are subject to cognitive biases. The Recognition-Primed Decision Model (RPDM) and the BRM are two examples of behavioral decision-making approaches.

<sup>3</sup><https://www.matt.hes.in.tum.de/pages/19gmopufr04wi>

## 7. Future work and conclusion

The RPDM is derived from the naturalistic decision-making framework that relies on mental simulations. The use of analogy for situation recognition and mental simulation for alternative evaluation and selection define the RPDM [140]. In RPDM, decision-makers (a) define the concerns, (b) recognize a concern based on previous knowledge and list the alternatives, (c) sequentially and iteratively apply mental simulation to check if an alternative addresses the concern, and (d) make the decision by choosing the first alternative that addresses the concern. The aim of the decision-maker while using RPDM is to find a “good-enough” alternative that meets an acceptability threshold.

The concept of bounded rationality was proposed by Simon in 1950s [14,141]. In BRM, decision-makers collect only a manageable subset of alternatives, rank the alternatives using heuristics, and choose a “satisficing” alternative (without using any optimization algorithm), which may or may not be the optimal one. The emphasis here is that, decision-makers look for the first workable option rather than the optimal option. Since the selection of an alternative neither maximizes nor satisfies, it is referred to as satisficing. For selecting a satisficing alternative, heuristics, for instance, could be *previous experience* or *team capabilities*.

Even though architects may not always be aware of how they make design decisions and their decision-making process may not be explicit [39], researchers have shown that architects implicitly apply one of the aforementioned models during decision-making. Moreover, the generic problem-solving methods such as 8Ds [142], GROW [143], PDCA [144], or OODA loop [13] can also aid architects structure the decision-making process. The Plan-Do-Check-Adjust (PDCA) is an iterative four-step problem-solving method used for continuous process improvement. The “plan-do-check” co-relates with the “hypothesis-experiment-evaluation” scientific method along with iteration cycles to ensure the alignment of the act phase with the goals or plans set for the iteration. Once the goals or objectives are established in the plan phase, do phase focuses on making small changes to meet the goals and to gather information on the effects of the changes. In the check phase, the information from the do phase is analyzed and finally, in the adjust/act phase the issues in the analysis results are recorded to improve the process. The issues identified in the adjust phase triggers the next iteration of the PDCA method.

The Observe, Orient, Decide, and Act (OODA) loop is a four-phase decision cycle used by strategists in many domains including business, litigation, and military strategy. The OODA

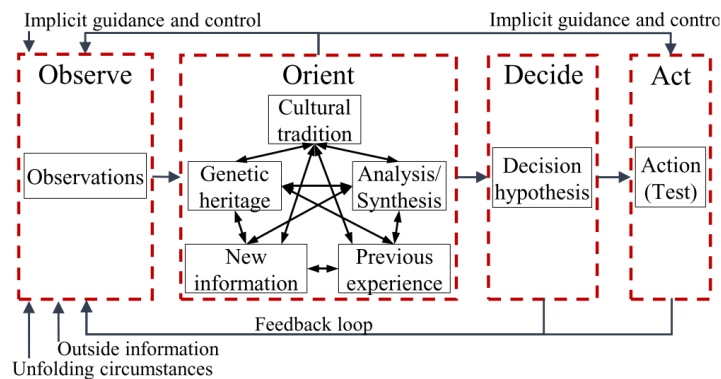


Figure 7.1: The OODA Loop decision cycle; Adapted from [13]

loop developed by Boyd is a generic decision-making process model. It describes how to gain competitive advantage in any situation. In software engineering, OODA loop has been discussed in the context of autonomous systems [145, 146]. Within the OODA loop, an actor makes *observations* about the surrounding environment, *orients* his or her thinking process by perceiving the important information based on the context, *decides* on a course of action, and finally *acts* on it. As shown in Figure 7.1, this process is iterative with loops providing feedback to the observe phase for constant reorientation and adaption.

Designing software systems involves strategic and tactical decision making while keeping in mind various factors such as long-term sustainability, technical capabilities of the teams, short-term availability of resources, and time constraints. In such a context, the OODA loop is relatable to architects and is an intuitive generic problem-solving method. To help architects remember the steps in DMMs, namely REM, RPDM and BRM, we have established the relationship between the DMMs and the OODA loop. It should be highlighted to the readers that we are in an early exploration phase of OODA loop’s applicability as an overarching framework. Furthermore, the applicability of OODA loop (or any other general problem-solving methods) to a certain decision-making stage (high-, medium-, or realization-level decisions as per our previous discussions) is open for investigation. However, we choose OODA loop for this discussion since, as explained in the next paragraphs, it enables us to structure thirty-three cognitive biases to the observe, orient, decide, and act phases during the decision-making process.

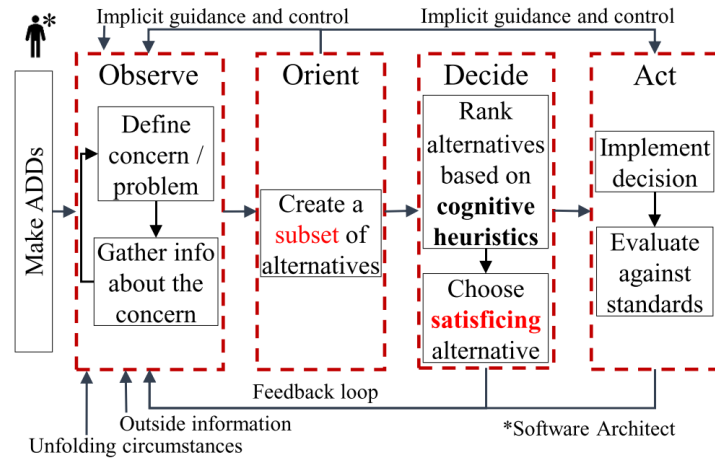


Figure 7.2: The BRM with the OODA Loop; BRM adapted from [14]

For example, as shown in Figure 7.2, architects define the design concerns in the *observe* phase and gather information related to those concerns until they feel it is complete. Then, architects create a subset of alternatives to *orient* themselves with the design concerns captured in the observe phase. Next, using heuristics such as previous experience and team capabilities, architects choose a “satisficing” alternative in the *decide* phase. Finally, the chosen alternative is implemented in the *act* phase. Similarly, we have also documented the relationship of REM and RPDM to the OODA loop in [133].

Furthermore, since architects are biased during their decision-making process, to raise awareness about cognitive biases during the ADM process, we also mapped the cognitive biases to the

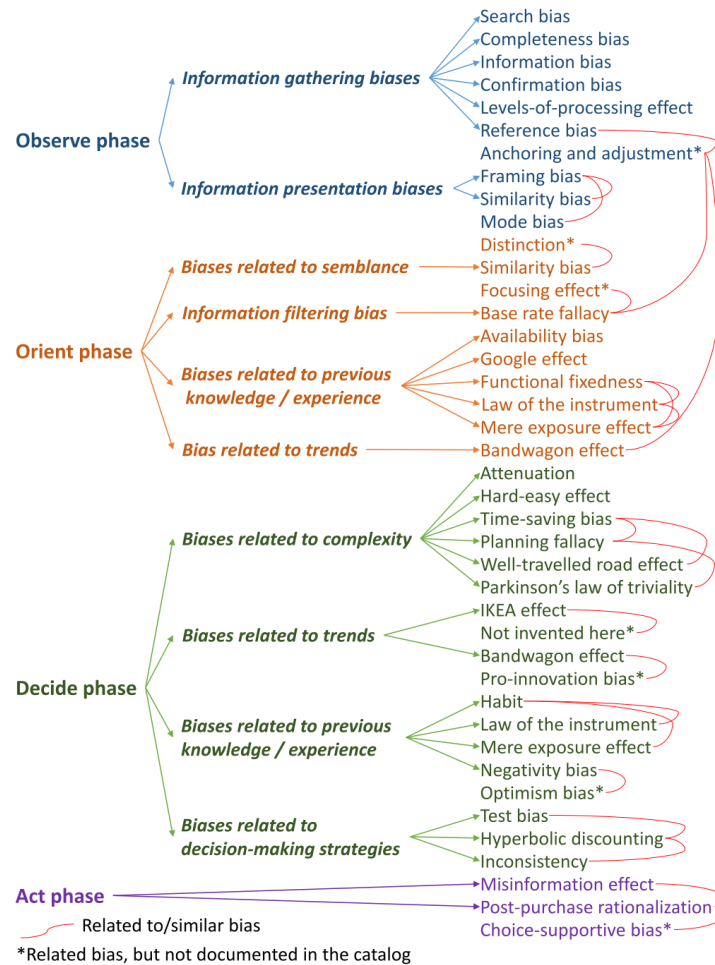


Figure 7.3: The two-stage classification of cognitive biases using the OODA loop

OODA loop. Even though more than two hundred cognitive biases have been identified by experts in various domains, we selected thirty-three biases that have already been discussed in the context of software engineering from publications including [147] and [148]. Furthermore, since the list of cognitive biases is extensive, a two-stage classification was used to modularize the information. In the first stage, each cognitive bias is assigned to one or more phases of the OODA loop. In the second stage, a further classification is made under each phase depending on the relationships between different biases. Figure 7.3 shows the two-stage hierarchical classification as well as relationships between biases (related to/similar). The selected thirty-three biases are represented as the leaf nodes<sup>4</sup>.

Within the observe phase, since architects focus on how to gather information and how to present them in the subsequent phase, biases can be classified into two main subcategories, namely, *information gathering biases* and *information presentation biases*. Information gathering biases include for instance, search, reference, and confirmation bias. On the other hand, framing and

<sup>4</sup><https://tum-master-thesis.herokuapp.com/>

similarity biases are classified as presentation biases. Furthermore, biases can be related to each another. For example, reference bias is related to anchoring and adjustment bias as they both establish a point of reference which sets the tone for further steps in decision making.

The orient phase consists of biases which influence how people interpret the information and orient themselves to the situation at hand. During this process, architects orient themselves by filtering available information and are influenced by the similarity of the situation, their previous experiences, and current trends. Hence, the subcategories under the orient phase are biases related to *semblance*, *information filtering biases*, *biases related to experience*, and *biases related to trends*.

Since the actual decision-making happens in the decide phase, this phase is associated with largest number of biases. The decisions are made based on the complexity of the problem, nature of how the solution will be identified (*trends*), *experience*, and *decision-making strategies*.

Corresponding to the act phase, since only two related biases, namely, misinformation effect and post-purchase rationalization were identified, we did not sub-classify these biases.

<b>Planning Fallacy</b>	
<b>Definitions Block</b>	
Definition 1: The tendency to underestimate task-completion times.	
Definition 2: It is a phenomenon in which predictions about how much time will be needed to complete a future task displays an optimism bias and underestimates the time needed.	
<b>OODA Class: Decide Phase</b>	<b>Subclass: Complexity</b>
<i>Reasoning for classification:</i> Time is a crucial factor in software projects. Often, the implementation times fall short of the initial estimates. The reason being underestimation of task-completion times due to lack of understanding of the complexities involved.	
<b>Examples and impact on architecture design decisions</b>	
<i>Example: Choosing spring-security as the security framework:</i> Spring is one of the most popular choice for developing Java-based enterprise applications. To meet the security requirements, spring-security would be an automatic choice as it is part of the framework itself. It is easy to assume that configuring the application security would be as easy as developing an application in spring. However, it is not an easy solution to implement without a proper understanding. If the decision-makers assumes that the security aspect is as easy as feature development, then it leads to an optimism bias resulting in time estimate errors.	
<i>Impact:</i> A common result is missing delivery deadlines. The added pressure resulting from the missed deadlines leads to implementation of sub-par solutions.	
<b>Debiasing techniques</b>	
The decision-maker must understand how to estimate time. There are many workflows for time estimation that can be used. One simple way is to add a buffer time to the initial time estimate in order to complete tasks. It is common to set the buffer time to 10% of the total estimate.	
<b>Related biases</b>	
Complexity bias, Parkinson's Law of triviality, Time-saving bias.	

Figure 7.4: An example of a cognitive bias (planning fallacy) as documented in the bias catalog

A catalog containing detailed information about each bias is presented in a web interface as well as documented in a report [132] as part of this research project. Readers are directed to that report for a detailed description regarding the selection of specific biases and the reasons for their classification under a specific category. The planning fallacy bias from the catalog is presented in Figure 7.4. Each bias is described using the same template containing (a) definitions from different sources, (b) OODA phase to which it belongs to, (c) the subclass within the classification, (d) rationale for classification under a specific class and subclass, (d) an example from the architectural decision making context, its implications, (e) hints on how architects can debias, and finally (f) their related biases.

In this research project we have explored three concepts (OODA loop, DMMs, and cognitive biases) along with their relationships in the context of ADM. To help architects relate their ADM process, be it rationalistic or naturalistic, we have mapped the steps in three different DMMs to the OODA loop which is a generic DMM. These three DMMs have already been discussed in the context of ADM by various researchers. Next, we have made the steps in those DMMs explicit using process flow diagrams and mapped those steps to the different phases of the OODA loop. Furthermore, since architects are biased during their ADM process, to raise awareness about cognitive biases we have documented thirty-three important biases as part of the cognitive bias catalog and mapped those biases to the OODA loop phases and subcategories.

Currently, we are investigating ways to improve the bias catalog and to make it accessible to architects and developers at large in our partner industry. It should be highlighted that even though the awareness of biases establishes the first point for debiasing, awareness alone is not sufficient. The best one can expect is a discussion on cognitive biases and some limited actions (as in the bias catalog) can be taken by architects to avoid those biases. Further research is required to provide substantial tool support to help architects debias during the ADM process.

### **A virtual meeting companion to support ADM in online meetings**

The work calendars of architects who are involved in multiple projects are typically cluttered with meetings. In these meetings, architects discuss design concerns, explore alternatives, and make design decisions. Due to time constraints and manual effort, such discussions are rarely documented. In weekly and bi-weekly meetings, architects even miss out on capturing the minutes of meetings. Overtime, architects due to cognitive limitations forget what decisions have been made in the previous meetings [36,149,150] and why those decisions were made [151,152].

Furthermore, in large software companies, software development teams are geographically distributed and architectural meetings are seldom face to face. The meetings are held virtually using online meeting tools such as Skype, Microsoft Teams, or Circuit. Over the last couple of decades, the verbal communication analysis using meeting support technology (speech recording, transcription, natural language understanding and generation) has become a significant area of research [153]. Given the context that architectural meetings in distributed teams are held online and many design decisions are made in those meetings, we believe that tapping into the conversations of architectural meetings can help to better understand and support architects during the ADM process in real time. Similar arguments have been discussed in the dissertation of Schiller [154]. The author presents a framework named STACHUS which is based on a grammar for meeting keywords including design problem, design decision, task etc. Once the



meetings are transcribed to text using an automatic speech recognition (ASR), the grammar is used to detect the meeting keywords and the meeting protocol is automatically generated. Furthermore, the system also pushes the identified tasks identified during the meetings to an issue management system along with the task description and its due date.

Our industry partner has started a research project titled “CircuitBot - A Virtual Meeting Companion” to support architects during architectural meetings. The meeting companion is referred to as CircuitBot since the tool Circuit is used primarily for online meetings. The high-level vision of the CircuitBot is shown in Figure 7.5.

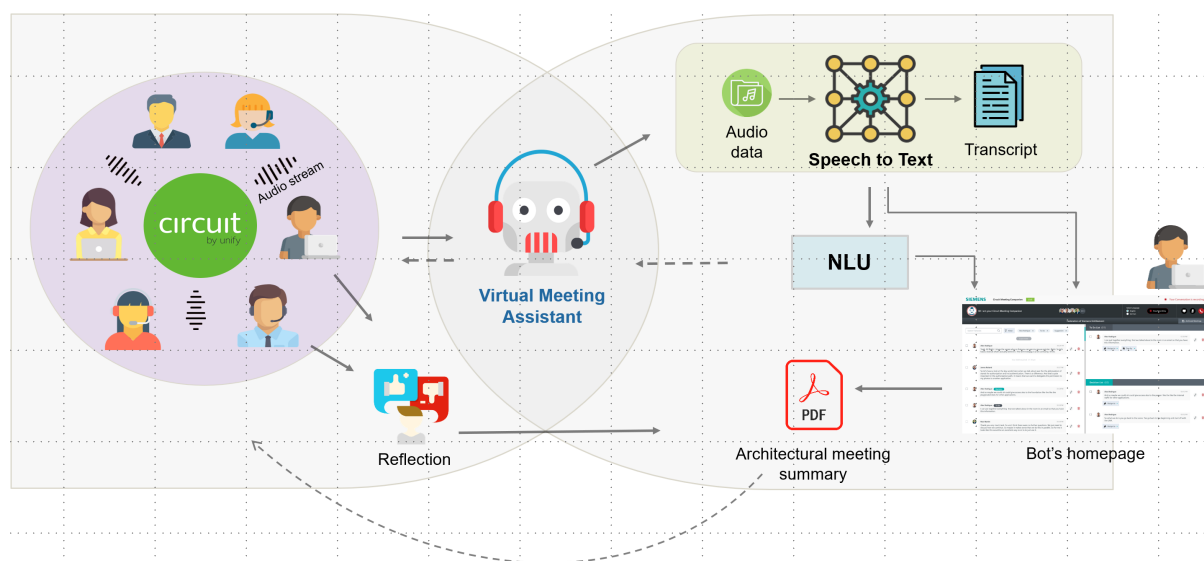


Figure 7.5: A virtual meeting companion for supporting architects in online meetings

The left side of Figure 7.5 shows the participants of the architectural meeting in a Circuit conference bridge. The moderator of the conference can invite the CircuitBot (Virtual Meeting Assistant in Figure 7.5) to join the conference. Once the bot joins the conference it starts to record the conversations and sends the audio chunks to the bot’s back-end for processing (shown on the right side of Figure 7.5). On receiving the audio chunks, the bot uses an ASR engine to transcribe the voice to text. The real-time transcripts of the conversations are presented to the meeting moderator on a web page. Moreover, the transcripts are sent to a natural language understanding (NLU) service which labels the transcripts for instance as action items, decisions, person responsible for an action item, and the deadline for the action item. These labels are also presented to the moderator on the bot’s homepage. The moderator has the controls to validate the recommendations from the NLU component and can finalize the meeting summary comprising of meeting meta-data, transcripts, labeled action items, decisions, and responsible individuals. The meeting summary in a pdf format is automatically pushed to the Circuit conversation so that the meeting participants can reflect on the discussions and update the meeting summary if required. The meeting summary is also indexed and persisted so as to enable look-up use cases. For instance, architects can ask the bot “when was a certain decision made?” or “what decisions were taken in the previous meeting?”.

Based on the idea presented above, we created a proof of concept (PoC) as part of a Master thesis project [155]. In this project, first, we conducted semi-structured interviews with ten practitioners to get their views on a virtual meeting assistant. Many of the interviewees suggested to include (a) action items, (b) assigned person, (c) task deadline, (d) decisions, and (e) catch words or keywords. In this PoC, we have used Speechmatics<sup>5</sup> as an ASR engine and Rasa<sup>6</sup> for NLU. Even though the evaluation results of the bot was not satisfactory (high word error rate for transcriptions and low F1 scores for decision detection), we were able to identify key challenges that need to be addressed to improve the meeting assistant and to support architects in online meetings. The key challenges include (a) training the ASR engine and Rasa NLU models with domain specific terms, (b) improving the quality of the voice received from the speakers, (c) considering emotions (e.g., pitch, pause, and speech rate) of the speaker to detect uncertainties, (d) including referring expressions to capture context, and (e) considering dialogue acts for improving decision knowledge extraction. During this study, we observed that the transcripts of the architectural meetings contain *dialogue acts* such as concern, alternatives, cons of an alternative, reiteration of a concern, the selection of an alternative (decision), and the justification for a decision. Considering such dialogue acts while processing the transcripts in combination with the grammar for meeting keywords proposed in the STACHUS framework can improve the design knowledge extraction process. We are currently investigating the aforementioned challenges in collaboration with our industry partner.

### 7.3 Conclusion

In this dissertation, we have presented a conceptual framework and its realization with the tool named ADeX for the automatic curation of design decisions and for supporting architects and developers during the decision-making process. We have documented in detail the research activities related to ADM based on a semi-systematic literature review. 227 publications which were considered for a detailed analysis and their documentation in Chapter 2 have helped us position our research and its relevance to the software architecture research community. The inputs received from our industry partner in combination with the findings from the literature review led to the formulation and prioritization of the use cases for ADeX.

We emphasize that forcing architects to document their decisions is often unreasonable and therefore, bottom-up approaches to AKM should complement the top-down approaches for capturing, structuring, and reusing AK in large software engineering projects. The distinguishing factor of ADeX as compared to the existing AKM tools that follow top-down approaches is that it does not expect stakeholders to manually capture data within AKM tools to support AKM use cases. ADeX uses NLP and ML techniques to identify design decisions made in the past by extracting information from disparate data sources (e.g., MS Project, Enterprise Architect, JIRA, and Github issues). As an important contribution of this dissertation, we have made the source code of ADeX as well as the datasets used for building the ML models publicly available.

Another major shortcoming of the existing AKM tools is the lack of their configurability to

---

<sup>5</sup><https://www.speechmatics.com/>

<sup>6</sup><https://rasa.com/>

align to different projects and organizational context. Hence, we have used a meta-model based system named SocioCortex as a foundation for ADeX. SocioCortex provides the flexibility to adapt the domain model (in our context, the AK model) at runtime and provides experts with the capability to define the AK model that meets specific project requirements.

Furthermore, we have presented a novel ontology-based approach to support architects and developers during ADM. By reusing the knowledge captured in a publicly available cross-domain ontology (the DBpedia ontology) we can identify architectural elements in natural language text and automatically generate alternative solutions that can be considered by decision makers during the ADM process. Furthermore, by combining both the ontology-based approach and a matrix-based approach, we have demonstrated how to quantify AK in an organization and how to identify those architects and developers (experts) who should be involved in ADM.

Finally, keeping in mind that the existing AKM tools are intrusive and add an overhead for architects, we made sure that the AK extracted by ADeX is presented to architects in a seamless integrated manner. Using different views that provide different perspectives, architects and developers can drill-down from a large amount of information collected from different sources and focus on specific aspects such as (a) quality attributes addressed or not addressed by design decisions, (b) architectural elements that are affected by design decisions, (c) experts who can help to make new design decisions, (d) similar decisions taken in the past, and (e) alternative solutions for ADM. We showed that ADeX can help software architects and developers, especially in large software projects, to answer the following questions:

- What design decisions have already been made?
- Which architectural elements and quality attributes are affected by a decision?
- Who should be involved in making a new decision?
- Which similar design decisions have been made?
- What are the alternatives to consider while making a decision?

Finally, the components and their respective views within ADeX have been developed over the last four years in collaboration with our industry partner who provided practical feedback. The components, as discussed in Chapter 6, have been evaluated both qualitatively and quantitatively. The lessons learned from those evaluation results have been summarized. Furthermore, we have also presented two specific ongoing activities, namely the influence of biases on ADM and supporting architects in online meetings. To further our research activities, we are actively seeking collaboration with researchers from cognitive science background who can provide better insights into the human aspects of architectural decision making.

Robillard et al. [156] argue that as a fuzzy human concept, recovering aspects related to design decisions can be notoriously difficult. The contribution of this dissertation should only be considered as a starting point for further exploring the application of recent trends in NLP and ML in the area of software architecture knowledge management. Moreover, given that, “software is designed by the people, for the people”, we also emphasize the need for focused research on the topics of human aspects of architectural decision making including cognitive biases and uncertainties during ADM.



---

## Bibliography

---

- [1] R. Capilla, A. Jansen, A. Tang, P. Avgeriou, and M. A. Babar, “10 years of software architecture knowledge management: Practice and future,” *Journal of Systems and Software*, vol. 116, pp. 191–205, 2016.
- [2] M. A. Babar, T. Dingsøy, P. Lago, and H. Van Vliet, *Software architecture knowledge management*. Springer, 2009.
- [3] A. Jansen and J. Bosch, “Software architecture as a set of architectural design decisions,” in *5th Working IEEE/IFIP Conference on Software Architecture (WICSA’05)*. IEEE, 2005, pp. 109–120.
- [4] ISO/IEC/IEEE, “Systems and software engineering – architecture description,” *ISO/IEC/IEEE 42010:2011(E)*, pp. 1–46, Dec 2011.
- [5] B. Kitchenham and S. Charters, “Guidelines for performing systematic literature reviews in software engineering,” Keele University and Durham University Joint Report, Tech. Rep. EBSE 2007-001, 2007.
- [6] T. Reschenhofer, M. Bhat, A. Hernandez-Mendez, and F. Matthes, “Lessons learned in aligning data and model evolution in collaborative information systems,” in *IEEE/ACM International Conference on Software Engineering Companion (ICSE-C)*. IEEE, 2016, pp. 132–141.
- [7] D. Ameller and X. Franch, “Ontology-based architectural knowledge representation: structural elements module,” in *International Conference on Advanced Information Systems Engineering*. Springer, 2011, pp. 296–301.
- [8] K. A. De Graaf, A. Tang, P. Liang, and H. Van Vliet, “Ontology-based software architecture documentation,” in *Joint Working IEEE/IFIP Conference on Software Architecture and European Conference on Software Architecture*. IEEE, 2012, pp. 121–130.
- [9] A. Tang, P. Liang, V. Clerc, and H. van Vliet, “Supporting co-evolving architectural requirements and design through traceability and reasoning,” *Relating Software Requirements and Software Architecture*, 2011.
- [10] O. C. S. Workgroup, “Oslc core specification version 2.0,” *Open Services for Lifecycle Col-*

laboration, *Technical Report*, 2010.

- [11] B. Bruegge and A. H. Dutoit, “Object-oriented software engineering. using uml, patterns, and java,” *Learning*, vol. 5, no. 6, p. 7, 2009.
- [12] F. Koch, “Rest-based data integration services for software engineering domain,” Master’s thesis, Technische Universität München, 2016.
- [13] G. Hammond, *The mind of war: John Boyd and American security*. Smithsonian Institution, 2012.
- [14] H. A. Simon, *The sciences of the artificial*. MIT press, 1996.
- [15] M. A. Babar, “Supporting the software architecture process with knowledge management,” in *Software Architecture Knowledge Management*. Springer, 2009, pp. 69–86.
- [16] J. S. van der Ven and J. Bosch, “Making the right decision - supporting architects with design decision data,” in *European Conference on Software Architecture*. Springer, 2013, pp. 176–183.
- [17] A. G. J. Jansen, *Architectural design decisions*. University Library of Groningen, 2008.
- [18] P. Kruchten, “An ontology of architectural design decisions in software intensive systems,” in *2nd Groningen Workshop on Software Variability*. Citeseer, 2004, pp. 54–61.
- [19] J. Bosch, “Software architecture: The next step,” in *European Workshop on Software Architecture*. Springer, 2004, pp. 194–199.
- [20] J. Tyree and A. Akerman, “Architecture decisions: Demystifying architecture,” *IEEE Software*, vol. 22, no. 2, pp. 19–27, 2005.
- [21] P. Kruchten, R. Capilla, and J. C. Dueñas, “The decision view’s role in software architecture practice,” *IEEE Software*, vol. 26, no. 2, pp. 36–42, 2009.
- [22] J. C. Dueñas and R. Capilla, “The decision view of software architecture,” in *European Workshop on Software Architecture*. Springer, 2005, pp. 222–230.
- [23] P. Clements, D. Garlan, L. Bass, J. Stafford, R. Nord, J. Ivers, and R. Little, *Documenting software architectures: views and beyond*. Pearson Education, 2002.
- [24] A. Tang, M. A. Babar, I. Gorton, and J. Han, “A survey of architecture design rationale,” *Journal of Systems and Software*, vol. 79, no. 12, pp. 1792–1804, 2006.
- [25] R. Farenhorst, J. F. Hoorn, P. Lago, and H. Van Vliet, “The lonesome architect,” in *Joint Working IEEE/IFIP Conference on Software Architecture & European Conference on Software Architecture. WICSA/ECISA 2009*. IEEE, 2009, pp. 61–70.
- [26] M. A. Babar and I. Gorton, “A tool for managing software architecture knowledge,” in *2nd Workshop on Sharing and Reusing Architectural Knowledge-Architecture, Rationale, and Design Intent (SHARK/ADI’07: ICSE Workshops 2007)*. IEEE, 2007, pp. 11–11.
- [27] M. Bhat, K. Shumaiev, A. Biesdorf, U. Hohenstein, M. Hassel, and F. Matthes, “Meta-model based framework for architectural knowledge management,” in *Proceedings of the*

- 
- 10th European Conference on Software Architecture Workshops*. ACM, 2016, p. 12.
- [28] C. Manteuffel, D. Tofan, H. Koziol, T. Goldschmidt, and P. Avgeriou, “Industrial implementation of a documentation framework for architectural decisions,” in *IEEE/IFIP Conference on Software Architecture (WICSA)*. IEEE, 2014, pp. 225–234.
- [29] G. Buchgeher and R. Weinreich, “Automatic tracing of decisions to architecture and implementation,” in *9th Working IEEE/IFIP Conference on Software Architecture (WICSA)*. IEEE, 2011, pp. 46–55.
- [30] R. Capilla, F. Nava, and C. Carrillo, “Effort estimation in capturing architectural knowledge,” in *Proceedings of the 2008 23rd IEEE/ACM International Conference on Automated Software Engineering*. IEEE Computer Society, 2008, pp. 208–217.
- [31] J. Lee, “Design rationale systems: understanding the issues,” *IEEE Expert*, vol. 12, no. 3, pp. 78–85, 1997.
- [32] L. Lee and P. Kruchten, “Capturing software architectural design decisions,” in *Canadian Conference on Electrical and Computer Engineering. CCECE 2007*. IEEE, 2007, pp. 686–689.
- [33] S. Ambler, *Agile modeling: effective practices for extreme programming and the unified process*. John Wiley & Sons, 2002.
- [34] C. J. Stettina and W. Heijstek, “Necessary and neglected?: an empirical study of internal documentation in agile software development teams,” in *Proceedings of the 29th ACM international conference on Design of communication*. ACM, 2011, pp. 159–166.
- [35] J. Sutherland, A. Viktorov, J. Blount, and N. Puntikov, “Distributed scrum: Agile project management with outsourced development teams,” in *40th Annual Hawaii International Conference on System Sciences. HICSS 2007*. IEEE, 2007, pp. 274a–274a.
- [36] C. Miesbauer and R. Weinreich, “Classification of design decisions—an expert survey in practice,” in *European Conference on Software Architecture*. Springer, 2013, pp. 130–145.
- [37] A. Kleebaum, J. O. Johanssen, B. Paech, R. Alkadhi, and B. Bruegge, “Decision knowledge triggers in continuous software engineering,” in *Proceedings of the 4th International Workshop on Rapid Continuous Software Engineering*. ACM, 2018, pp. 23–26.
- [38] S. T. Hassard, A. Blandford, and A. L. Cox, “Analogies in design decision-making,” in *Proceedings of the 23rd British HCI Group Annual Conference on People and Computers: Celebrating People and Technology*. British Computer Society, 2009, pp. 140–148.
- [39] A. Tang, M. Razavian, B. Paech, and T.-M. Hesse, “Human aspects in software architecture decision making: a literature review,” in *IEEE International Conference on Software Architecture (ICSA)*. IEEE, 2017, pp. 107–116.
- [40] C. Zannier and F. Maurer, “Social factors relevant to capturing design decisions,” in *Proceedings of the Second Workshop on SHaring and Reusing architectural Knowledge Architecture, Rationale, and Design Intent*. IEEE Computer Society, 2007, p. 1.
- [41] C. Zannier, M. Chiasson, and F. Maurer, “A model of design decision making based on em-

- pirical results of interviews with software designers,” *Information and Software Technology*, vol. 49, no. 6, pp. 637–653, 2007.
- [42] A. H. Dutoit, R. McCall, I. Mistrik, and B. Paech, “Rationale management in software engineering: Concepts and techniques,” in *Rationale Management in Software Engineering*. Springer, 2006, pp. 1–48.
- [43] A. H. Dutoit, R. McCall, I. Mistrik, and B. Paech, *Rationale management in software engineering*. Springer Science & Business Media, 2007.
- [44] 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, pp. 918–934, 2007.
- [45] J. E. Burge, J. M. Carroll, R. McCall, and I. Mistrik, *Rationale-based software engineering*. Springer, 2008.
- [46] J. Cleland-Huang, R. Settini, X. Zou, and P. Solc, “Automated classification of non-functional requirements,” *Requirements Engineering*, vol. 12, no. 2, pp. 103–120, 2007.
- [47] J. Slankas and L. Williams, “Automated extraction of non-functional requirements in available documentation,” in *1st International Workshop on Natural Language Analysis in Software Engineering (NaturaLiSE)*. IEEE, 2013, pp. 9–16.
- [48] L. Chung, B. A. Nixon, E. Yu, and J. Mylopoulos, *Non-functional requirements in software engineering*. Springer Science & Business Media, 2012, vol. 5.
- [49] A. Mockus and J. D. Herbsleb, “Expertise browser: a quantitative approach to identifying expertise,” in *Proceedings of the 24rd International Conference on Software Engineering. ICSE 2002*. IEEE, 2002, pp. 503–512.
- [50] A. Kleebaum, J. O. Johanssen, B. Paech, and B. Bruegge, “How do practitioners manage decision knowledge during continuous software engineering?” in *Proceedings of the 31st International Conference on Software Engineering and Knowledge Engineering, ser. SEKE’19*. KSI Research Inc., 2019, pp. 735–740.
- [51] J. O. Johanssen, A. Kleebaum, B. Bruegge, and B. Paech, “Towards a systematic approach to integrate usage and decision knowledge in continuous software engineering,” in *CSE@ SE*, 2017, pp. 7–11.
- [52] A. Kleebaum, J. O. Johanssen, B. Paech, and B. Bruegge, “Tool support for decision and usage knowledge in continuous software engineering,” in *Proceedings of the 3rd Workshop on Continuous Software Engineering*. Ulm, Germany: CEUR-WS.org, 2018, pp. 74–77.
- [53] T.-M. Hesse and B. Paech, “Supporting the collaborative development of requirements and architecture documentation,” in *3rd International Workshop on the Twin Peaks of Requirements and Architecture (TwinPeaks)*. IEEE, 2013, pp. 22–26.
- [54] T.-M. Hesse, A. Kuehlwein, and T. Roehm, “Decdoc: A tool for documenting design decisions collaboratively and incrementally,” in *1st International Workshop on Decision Making in Software ARCHitecture (MARCH)*. IEEE, 2016, pp. 30–37.
- [55] R. Alkadhi, J. O. Johanssen, E. Guzman, and B. Bruegge, “React: an approach for capturing



- rationale in chat messages,” in *ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*. IEEE, 2017, pp. 175–180.
- [56] R. M. A. Alkadhi, “Rationale in developers’ communication,” Ph.D. dissertation, Technische Universität München, 2018.
- [57] F. Gilson and D. Weyns, “When natural language processing jumps into collaborative software engineering,” in *IEEE International Conference on Software Architecture Companion (ICSA-C)*. IEEE, 2019, pp. 238–241.
- [58] A. Kleebaum, M. Konersmann, M. Langhammer, B. Paech, M. Goedicke, and R. Reussner, “Continuous design decision support,” in *Managed Software Evolution*. Springer, 2019, pp. 107–139.
- [59] H. van Vliet and A. Tang, “Decision making in software architecture,” *Journal of Systems and Software*, vol. 117, pp. 638–644, 2016.
- [60] M. Bhat, K. Shumaiev, A. Biesdorf, U. Hohenstein, and F. Matthes, “Automatic extraction of design decisions from issue management systems: a machine learning based approach,” in *European Conference on Software Architecture*. Springer, 2017, pp. 138–154.
- [61] M. Bhat, K. Shumaiev, and F. Matthes, “Towards a framework for managing architectural design decisions,” in *Proceedings of the 11th European Conference on Software Architecture: Companion Proceedings*. ACM, 2017, pp. 48–51.
- [62] M. Bhat, K. Shumaiev, A. Biesdorf, U. Hohenstein, M. Hassel, and F. Matthes, “An ontology-based approach for software architecture recommendations,” in *23rd Americas Conference on Information Systems (AMCIS), Boston, MA, USA, 2017*. [Online]. Available: <http://aisel.aisnet.org/amcis2017/SemanticsIS/Presentations/7>
- [63] M. Bhat, K. Shumaiev, K. Koch, U. Hohenstein, A. Biesdorf, and F. Matthes, “An expert recommendation system for design decision making: Who should be involved in making a design decision?” in *2018 IEEE International Conference on Software Architecture (ICSA)*. IEEE, 2018, pp. 85–8509.
- [64] M. Bhat, C. Tinnes, K. Shumaiev, , A. Biesdorf, U. Hohenstein, and F. Matthes, “Adex: A tool for automatic curation of design decision knowledge for architectural decision recommendations,” in *2019 IEEE International Conference on Software Architecture Companion (ICSA-C)*. IEEE, 2019.
- [65] J. Biolchini, P. G. Mian, A. C. C. Natali, and G. H. Travassos, “Systematic review in software engineering,” *System Engineering and Computer Science Department COPPE/UFRJ, Technical Report ES*, vol. 679, no. 05, p. 45, 2005.
- [66] D. Budgen and P. Brereton, “Performing systematic literature reviews in software engineering,” in *Proceedings of the 28th international conference on Software engineering*. ACM, 2006, pp. 1051–1052.
- [67] T. Dyba, T. Dingsoyr, and G. K. Hanssen, “Applying systematic reviews to diverse study types: An experience report,” in *1st International Symposium on Empirical Software Engineering and Measurement. ESEM 2007*. IEEE, 2007, pp. 225–234.

- [68] B. Kitchenham, "Procedures for performing systematic reviews," *Keele, UK, Keele University*, vol. 33, no. 2004, pp. 1–26, 2004.
- [69] K. Shumaiev and M. Bhat, "Automatic uncertainty detection in software architecture documentation," in *IEEE International Conference on Software Architecture Workshops (ICSAW)*. IEEE, 2017, pp. 216–219.
- [70] K. Shumaiev, M. Bhat, O. Klymenko, A. Biesdorf, U. Hohenstein, and F. Matthes, "Uncertainty expressions in software architecture group decision making: explorative study," in *Proceedings of the 12th European Conference on Software Architecture: Companion Proceedings*. ACM, 2018, p. 42.
- [71] D. Kahneman, *Thinking, fast and slow*. Macmillan, 2011.
- [72] A. Brown and G. Wilson, "The architecture of open source applications, volume i and volume ii," *Ebook, May*, 2012.
- [73] H. Rittel, "On the planning crisis: Systems analysis of the 'first and second generations'," *Bedriftskonomen*, vol. 8, pp. 390–396, 1972.
- [74] H. W. Rittel and M. M. Webber, "Dilemmas in a general theory of planning," *Policy Sciences*, vol. 4, no. 2, pp. 155–169, 1973.
- [75] 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.
- [76] P. Liang and P. Avgeriou, "Tools and technologies for architecture knowledge management," in *Software Architecture Knowledge Management*. Springer, 2009, pp. 91–111.
- [77] P. Kruchten, P. Lago, H. Van Vliet, and T. Wolf, "Building up and exploiting architectural knowledge," in *5th Working IEEE/IFIP Conference on Software Architecture. WICSA 2005*. IEEE, 2005, pp. 291–292.
- [78] J. Xu, "Improving the usability of an integrated decision support system for design decision making," Master's thesis, Technische Universität München, 2017.
- [79] M. A. Babar, I. Gorton, and B. Kitchenham, "A framework for supporting architecture knowledge and rationale management," in *Rationale Management in Software Engineering*. Springer, 2006, pp. 237–254.
- [80] P. Lago, R. Farenhorst, P. Avgeriou, R. C. de Boer, V. Clerc, A. Jansen, and H. van Vliet, "The griffin collaborative virtual community for architectural knowledge management," in *Collaborative Software Engineering*. Springer, 2010, pp. 195–217.
- [81] G. Fischer and J. Otswald, "Knowledge management: problems, promises, realities, and challenges," *IEEE Intelligent Systems*, vol. 16, no. 1, pp. 60–72, 2001.
- [82] J. S. Van Der Ven, A. Jansen, P. Avgeriou, and D. K. Hammer, *Using architectural decisions*. University of Groningen, Johann Bernoulli Institute for Mathematics and Computer Science, 2006.

- 
- [83] W. C. Regli, X. Hu, M. Atwood, and W. Sun, "A survey of design rationale systems: approaches, representation, capture and retrieval," *Engineering with Computers*, vol. 16, no. 3-4, pp. 209–235, 2000.
- [84] M. A. Babar, A. Northway, I. Gorton, P. Heuer, and T. Nguyen, "Introducing tool support for managing architectural knowledge: an experience report," in *15th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems (ECBS)*. IEEE, 2008, pp. 105–113.
- [85] P. Haumer, K. Pohl, K. Weidenhaupt, and M. Jarke, "Improving reviews by extended traceability," in *Proceedings of the 32nd Annual Hawaii International Conference on Systems Sciences. HICSS-32*. IEEE, 1999, pp. 10–pp.
- [86] A. MacLean, R. M. Young, V. M. Bellotti, and T. P. Moran, "Questions, options, and criteria: Elements of design space analysis," *Human-Computer Interaction*, vol. 6, no. 3-4, pp. 201–250, 1991.
- [87] B. Ramesh and M. Jarke, "Toward reference models for requirements traceability," *IEEE Transactions on Software Engineering*, no. 1, pp. 58–93, 2001.
- [88] E. J. Conklin and K. B. Yakemovic, "A process-oriented approach to design rationale," *Human-Computer Interaction*, vol. 6, no. 3-4, pp. 357–391, 1991.
- [89] J. Daiber, M. Jakob, C. Hokamp, and P. N. Mendes, "Improving efficiency and accuracy in multilingual entity extraction," in *Proceedings of the 9th International Conference on Semantic Systems*. ACM, 2013, pp. 121–124.
- [90] P. Liang, A. Jansen, and P. Avgeriou, "Collaborative software architecting through architectural knowledge sharing," *Collaborative Software Engineering (CoSE)*, pp. 343–368, 2008.
- [91] N. Schuster, "Adkwik—a collaborative system for architectural decision modeling and decision process support based on web 2.0 technologies," *Stuttgart Media University*, 2007.
- [92] A. Jansen, J. Van Der Ven, P. Avgeriou, and D. K. Hammer, "Tool support for architectural decisions," in *The Working IEEE/IFIP Conference on Software Architecture. WICSA'07*. Ieee, 2007, pp. 4–4.
- [93] L. Lee and P. Kruchten, "A tool to visualize architectural design decisions," in *International Conference on the Quality of Software Architectures*. Springer, 2008, pp. 43–54.
- [94] M. Shahin, P. Liang, and M. R. Khayyambashi, "A survey of architectural design decision models and tools," *Technical Report SBU-RUG-2009-SL-01, Sheikh Bahaei University & University of Groningen*, 2009.
- [95] F. Matthes, C. Neubert, and A. Steinhoff, "Hybrid wikis: Empowering users to collaboratively structure information," *ICSOFT (1)*, vol. 11, pp. 250–259, 2011.
- [96] R. B. Grady and D. L. Caswell, *Software metrics: establishing a company-wide program*. Prentice-Hall, Inc., 1987.
- [97] R. B. Grady, *Practical software metrics for project management and process improvement*. Prentice-Hall, Inc., 1992.

- [98] T. Büchner, F. Matthes, and C. Neubert, “Data model driven implementation of web co-operation systems with tricia,” in *Objects and Databases*. Springer, 2010, pp. 70–84.
- [99] A. Tang, P. Liang, and H. Van Vliet, “Software architecture documentation: The road ahead,” in *9th Working IEEE/IFIP Conference on Software Architecture*. IEEE, 2011, pp. 252–255.
- [100] A. Osterwalder, “The business model ontology a proposition in a design science approach,” Ph.D. dissertation, Université de Lausanne, Faculté des hautes études commerciales, 2004.
- [101] A. Osterwalder and Y. Pigneur, *Business model generation: a handbook for visionaries, game changers, and challengers*. John Wiley & Sons, 2010.
- [102] C. Hofmeister, R. Nord, and D. Soni, *Applied software architecture*. Addison-Wesley Professional, 2000.
- [103] T. Reschenhofer, I. Monahov, and F. Matthes, “Type-safety in ea model analysis,” in *IEEE 18th International Enterprise Distributed Object Computing Conference Workshops and Demonstrations (EDOCW)*. IEEE, 2014, pp. 87–94.
- [104] P. Browne, *JBoss Drools business rules*. Packt Publishing Ltd, 2009.
- [105] F. Galiegue, K. Zyp *et al.*, “Json schema: Core definitions and terminology,” *Internet Engineering Task Force (IETF)*, p. 32, 2013.
- [106] P. Bagrecha, “Implementation of an exploratory workbench for identifying similar design decisions,” Master’s thesis, Technische Universität München, 2018.
- [107] E. Prud, A. Seaborne *et al.*, “Sparql query language for rdf,” *W3C recommendation*, 2006.
- [108] A. Jena, “A free and open source java framework for building semantic web and linked data applications,” *Available online: jena.apache.org (accessed on 28 April 2015)*, 2015.
- [109] R. Kazman, M. Klein, and P. Clements, “Atam: Method for architecture evaluation,” Carnegie-Mellon Univ Pittsburgh PA Software Engineering Institute, Tech. Rep., 2000.
- [110] I. Hussain, L. Kosseim, and O. Ormandjieva, “Using linguistic knowledge to classify non-functional requirements in srs documents,” in *International Conference on Application of Natural Language to Information Systems*. Springer, 2008, pp. 287–298.
- [111] J. Cleland-Huang, R. Settimi, X. Zou, and P. Solc, “The detection and classification of non-functional requirements with application to early aspects,” in *14th IEEE International Requirements Engineering Conference (RE’06)*. IEEE, 2006, pp. 39–48.
- [112] L. Rosenhainer, “Identifying crosscutting concerns in requirements specifications,” in *Proceedings of OOPSLA Early Aspects*, 2004.
- [113] A. Casamayor, D. Godoy, and M. Campo, “Identification of non-functional requirements in textual specifications: A semi-supervised learning approach,” *Information and Software Technology*, vol. 52, no. 4, pp. 436–445, 2010.
- [114] T. Menzies, B. Caglayan, E. Kocaguneli, J. Krall, F. Peters, and B. Turhan, “The promise repository of empirical software engineering data,” 2012. [Online]. Available:

---

<http://promisedata.org/repository>

- [115] W. Zhang, Y. Yang, Q. Wang, and F. Shu, “An empirical study on classification of non-functional requirements,” in *23rd International Conference on Software Engineering and Knowledge Engineering (SEKE 2011)*, 2011, pp. 190–195.
- [116] Z. Kurtanović and W. Maalej, “Automatically classifying functional and non-functional requirements using supervised machine learning,” in *IEEE 25th International Requirements Engineering Conference (RE)*. IEEE, 2017, pp. 490–495.
- [117] K. Koch, “Design and implementation of an expert recommendation system for making design decisions,” Master’s thesis, Technische Universität München, 2017.
- [118] P. Bagrecha, “Implementation of an exploratory workbench for identifying similar design decisions,” Master’s thesis, Technische Universität München, 2017.
- [119] M. Ruppel, “Automatic extraction of design decision relationships from a task management system,” Master’s thesis, Technische Universität München, 2017.
- [120] C. J. Van Rijsbergen, “Information retrieval,” 1979. [Online]. Available: [www.onlinelibrary.wiley.com](http://www.onlinelibrary.wiley.com)
- [121] Caphyon, “Ctr study,” <https://www.advancedwebanking.com/cloud/ctrstudy/>, 2016, accessed: 2018-09-13.
- [122] M. F. Porter, “An algorithm for suffix stripping,” *Program*, vol. 14, no. 3, pp. 130–137, 1980.
- [123] P. Refaeilzadeh, L. Tang, and H. Liu, “Cross-validation,” in *Encyclopedia of Database Systems*. Springer, 2009, pp. 532–538.
- [124] C.-C. Chang and C.-J. Lin, “Libsvm: a library for support vector machines,” *ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 2, no. 3, p. 27, 2011.
- [125] X. Meng, J. Bradley, B. Yavuz, E. Sparks, S. Venkataraman, D. Liu, J. Freeman, D. Tsai, M. Amde, S. Owen *et al.*, “Mllib: Machine learning in apache spark,” *The Journal of Machine Learning Research*, vol. 17, no. 1, pp. 1235–1241, 2016.
- [126] C. D. Manning, C. D. Manning, and H. Schütze, *Foundations of statistical natural language processing*. MIT press, 1999.
- [127] M. Razavian, A. Tang, R. Capilla, and P. Lago, “In two minds: how reflections influence software design thinking,” *Journal of Software: Evolution and Process*, vol. 28, no. 6, pp. 394–426, 2016.
- [128] I. Groher and R. Weinreich, “A study on architectural decision-making in context,” in *12th Working IEEE/IFIP Conference on Software Architecture (WICSA)*. IEEE, 2015, pp. 11–20.
- [129] R. C. De Boer, R. Farenhorst, P. Lago, H. Van Vliet, V. Clerc, and A. Jansen, “Architectural knowledge: Getting to the core,” in *International Conference on the Quality of Software Architectures*. Springer, 2007, pp. 197–214.

- [130] K. Telschig, N. Schöffel, K.-B. Schultis, C. Elsner, and A. Knapp, “Seco patterns: Architectural decision support in software ecosystems,” in *1st International Workshop on Decision Making in Software ARCHitecture (MARCH)*. IEEE, 2016, pp. 38–44.
- [131] A. R. Hevner, “A three cycle view of design science research,” *Scandinavian Journal of Information Systems*, vol. 19, no. 2, p. 4, 2007.
- [132] A. Manjunath, “Decision-making and cognitive biases in designing software architectures,” Master’s thesis, Technische Universität München, 2018.
- [133] A. Manjunath, M. Bhat, K. Shumaiev, A. Biesdorf, and F. Matthes, “Decision making and cognitive biases in designing software architectures,” in *IEEE International Conference on Software Architecture Companion (ICSA-C)*. IEEE, 2018, pp. 52–55.
- [134] G. A. Klein, *Streetlights and shadows: Searching for the keys to adaptive decision making*. MIT Press, 2011.
- [135] M. Crowder, “Decision-making: two schools of thought or one?” in *Salford Postgraduate Annual Research Conference*, 2012, p. 47.
- [136] M. Crowder, “Decision-making in practice: The use of cognitive heuristics by senior managers,” 2013. [Online]. Available: <http://hdl.handle.net/10034/314940>
- [137] E. Brunswik, “The conceptual framework of psychology,” *Psychological Bulletin*, vol. 49, no. 6, pp. 654–656, 1952.
- [138] D. J. Snowden and M. E. Boone, “A leader’s framework for decision making,” *Harvard Business Review*, vol. 85, no. 11, p. 68, 2007.
- [139] W. Edwards, “The theory of decision making,” *Psychological Bulletin*, vol. 51, no. 4, p. 380, 1954.
- [140] C. M. Mitchell, J. G. Morris, J. J. Ockerman, and W. J. Potter, “Recognition primed decision making as a technique to support reuse in software design,” *Naturalistic Decision Making*, pp. 305–318, 1997.
- [141] H. A. Simon, “Models of man; social and rational.” *Wiley*, 1957. [Online]. Available: <https://psycnet.apa.org/record/1958-00363-000>
- [142] L. Rambaud, *8D structured problem solving: A guide to creating high quality 8D reports*. Phred Solutions, 2006.
- [143] M. Landsberg, *The Tao of coaching: Boost your effectiveness at work by inspiring and developing those around you*. Profile Books, 2015.
- [144] N. R. Tague, *The quality toolbox*. ASQ Quality Press Milwaukee, WI, 2005, vol. 600.
- [145] S. Behere and M. Torngren, “A functional architecture for autonomous driving,” in *1st International Workshop on Automotive Software Architecture (WASA)*. IEEE, 2015, pp. 3–10.
- [146] S. Karim and C. Heinze, “Experiences with the design and implementation of an agent-based autonomous uav controller,” in *Proceedings of the 4th International Joint Conference*

- 
- on Autonomous Agents and Multiagent Systems*. ACM, 2005, pp. 19–26.
- [147] A. Zalewski, K. Borowa, and A. Ratkowski, “On cognitive biases in architecture decision making,” in *European Conference on Software Architecture*. Springer, 2017, pp. 123–137.
- [148] W. Stacy and J. MacMillan, “Cognitive bias in software engineering,” *Communications of the ACM*, vol. 38, no. 6, pp. 57–63, 1995.
- [149] S. Rugaber, S. B. Ornburn, and R. J. LeBlanc, “Recognizing design decisions in programs,” *IEEE Software*, vol. 7, no. 1, pp. 46–54, 1990.
- [150] K. Power and R. Wirfs-Brock, “Understanding architecture decisions in context,” in *European Conference on Software Architecture*. Springer, 2018, pp. 284–299.
- [151] C. Potts and G. Bruns, “Recording the reasons for design decisions,” in *Proceedings of the 10th international conference on Software engineering*. IEEE Computer Society Press, 1988, pp. 418–427.
- [152] G. Pedraza-Garcia, H. Astudillo, and D. Correal, “Analysis of design meetings for understanding software architecture decisions,” in *XL Latin American Computing Conference (CLEI)*. IEEE, 2014, pp. 1–10.
- [153] A. Popescu-Belis, D. Lalanne, and H. Bourlard, “Finding information in multimedia records of meetings,” Idiap, Tech. Rep., 2011.
- [154] J. Schiller, “A framework for externalizing information in agile meetings,” Ph.D. dissertation, Technische Universität München, 2011. [Online]. Available: <http://nbn-resolving.de/urn:nbn:de:bvb:91-diss-20110531-998198-1-0>
- [155] O. Klymenko, “Automatic documentation of results during online architectural meetings,” Master’s thesis, Technische Universität München, 2019.
- [156] M. P. Robillard, A. Marcus, C. Treude, G. Bavota, O. Chaparro, N. Ernst, M. A. Gerosa, M. Godfrey, M. Lanza, M. Linares-Vásquez *et al.*, “On-demand developer documentation,” in *IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 2017, pp. 479–483.

---

## Primary studies used in literature review

---

- [P1] P. Kruchten, P. Lago, H. Van Vliet, and T. Wolf, “Building up and exploiting architectural knowledge,” in *5th Working IEEE/IFIP Conference on Software Architecture. WICSA 2005*. IEEE, 2005, pp. 291–292.
- [P2] P. Kruchten, P. Lago, and H. Van Vliet, “Building up and reasoning about architectural knowledge,” in *International Conference on the Quality of Software Architectures*. Springer, 2006, pp. 43–58.
- [P3] D. L. Parnas and P. C. Clements, “A rational design process: How and why to fake it,” *IEEE Transactions on Software Engineering*, no. 2, pp. 251–257, 1986.
- [P4] D. B. Walz, J. J. Elam, and B. Curtis, “Inside a software design team: knowledge acquisition, sharing, and integration,” *Communications of the ACM*, vol. 36, no. 10, pp. 63–77, 1993.
- [P5] M. Lavallée and P. N. Robillard, “Causes of premature aging during software development: an observational study,” in *Proceedings of the 12th International Workshop on Principles of Software Evolution and the 7th annual ERCIM Workshop on Software Evolution*. ACM, 2011, pp. 61–70.
- [P6] J. S. van der Ven, A. G. Jansen, J. A. Nijhuis, and J. Bosch, “Design decisions: The bridge between rationale and architecture,” in *Rationale Management in Software Engineering*. Springer, 2006, pp. 329–348.
- [P7] R. Farenhorst, J. F. Hoorn, P. Lago, and H. Van Vliet, “The lonesome architect,” in *Joint Working IEEE/IFIP Conference on Software Architecture & European Conference on Software Architecture. WICSA/ECSCA 2009*. IEEE, 2009, pp. 61–70.
- [P8] R. Farenhorst, P. Lago, and H. Van Vliet, “Effective tool support for architectural knowledge sharing,” in *European Conference on Software Architecture*. Springer, 2007, pp. 123–138.
- [P9] R. Farenhorst, R. Izaks, P. Lago, and H. Van Vliet, “A just-in-time architectural knowledge sharing portal,” in *7th Working IEEE/IFIP Conference on Software Architecture. WICSA 2008*. IEEE, 2008, pp. 125–134.



- [P10] R. Farenhorst and H. Van Vliet, "Understanding how to support architects in sharing knowledge," in *ICSE Workshop on Sharing and Reusing Architectural Knowledge. SHARK'09*. IEEE, 2009, pp. 17–24.
- [P11] W. Bu, A. Tang, and J. Han, "An analysis of decision-centric architectural design approaches," in *Proceedings of the 2009 ICSE Workshop on Sharing and Reusing Architectural Knowledge*. IEEE Computer Society, 2009, pp. 33–40.
- [P12] P. Liang, A. Jansen, and P. Avgeriou, "Selecting a high-quality central model for sharing architectural knowledge," in *8th International Conference on Quality Software. QSIC'08*. IEEE, 2008, pp. 357–365.
- [P13] H. Van Vliet, "Software architecture knowledge management," in *19th Australian Conference on Software Engineering. ASWEC 2008*. IEEE, 2008, pp. 24–31.
- [P14] R. C. De Boer and R. Farenhorst, "In search of architectural knowledge," in *Proceedings of the 3rd international workshop on Sharing and reusing architectural knowledge*. ACM, 2008, pp. 71–78.
- [P15] R. C. De Boer and H. Van Vliet, "On the similarity between requirements and architecture," *Journal of Systems and Software*, vol. 82, no. 3, pp. 544–550, 2009.
- [P16] F. Chen, "From architecture to requirements: Relating requirements and architecture for better requirements engineering," in *IEEE 22nd International Requirements Engineering Conference (RE)*. IEEE, 2014, pp. 451–455.
- [P17] A. Jansen, P. Avgeriou, and J. S. van der Ven, "Enriching software architecture documentation," *Journal of Systems and Software*, vol. 82, no. 8, pp. 1232–1248, 2009.
- [P18] O. Zimmermann, L. Wegmann, H. Koziolok, and T. Goldschmidt, "Architectural decision guidance across projects-problem space modeling, decision backlog management and cloud computing knowledge," in *12th Working IEEE/IFIP Conference on Software Architecture (WICSA)*. IEEE, 2015, pp. 85–94.
- [P19] R. C. de Boer, "Archimedes publication and integration of architectural knowledge," in *IEEE International Conference on Software Architecture Workshops (ICSAW)*. IEEE, 2017, pp. 268–271.
- [P20] 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.
- [P21] Z. Li, P. Liang, and P. Avgeriou, "Application of knowledge-based approaches in software architecture: A systematic mapping study," *Information and Software Technology*, vol. 55, no. 5, pp. 777–794, 2013.
- [P22] R. Weinreich and I. Groher, "A fresh look at codification approaches for sakm: A systematic literature review," in *European Conference on Software Architecture*. Springer, 2014, pp. 1–16.
- [P23] M. S. Nejad, S. Moaven, J. Habibi, and R. Alidousti, "Toward a collaborative method

- for knowledge management of software architectural decisions based on trust,” in *12th International Conference on Fuzzy Systems and Knowledge Discovery (FSKD)*. IEEE, 2015, pp. 828–834.
- [P24] T.-M. Hesse and B. Paech, “Documenting relations between requirements and design decisions: A case study on design session transcripts,” in *International Working Conference on Requirements Engineering: Foundation for Software Quality*. Springer, 2016, pp. 188–204.
- [P25] M. Soliman, M. Galster, A. R. Salama, and M. Riebisch, “Architectural knowledge for technology decisions in developer communities: An exploratory study with stack-overflow,” in *13th Working IEEE/IFIP Conference on Software Architecture (WICSA)*. IEEE, 2016, pp. 128–133.
- [P26] M. Soliman, M. Galster, and M. Riebisch, “Developing an ontology for architecture knowledge from developer communities,” in *IEEE International Conference on Software Architecture (ICSA)*. IEEE, 2017, pp. 89–92.
- [P27] T. D. LaToza, E. Shabani, and A. Van Der Hoek, “A study of architectural decision practices,” in *6th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE), 2013*. IEEE, 2013, pp. 77–80.
- [P28] J. Musil, F. J. Ekaputra, M. Sabou, T. Ionescu, D. Schall, A. Musil, and S. Biffi, “Continuous architectural knowledge integration: Making heterogeneous architectural knowledge available in large-scale organizations,” in *2017 IEEE International Conference on Software Architecture (ICSA)*. IEEE, 2017, pp. 189–192.
- [P29] R. Capilla, A. Jansen, A. Tang, P. Avgeriou, and M. A. Babar, “10 years of software architecture knowledge management: Practice and future,” *Journal of Systems and Software*, vol. 116, pp. 191–205, 2016.
- [P30] M. Sintzoff, “Suggestions for composing and specifying program design decisions,” in *International Symposium on Programming*. Springer, 1980, pp. 311–326.
- [P31] S. Rugaber, S. B. Ornburn, and R. J. LeBlanc, “Recognizing design decisions in programs,” *IEEE Software*, vol. 7, no. 1, pp. 46–54, 1990.
- [P32] A. Cimitile, F. Lanubile, and G. Visaggio, “Traceability based on design decisions,” in *Proceedings Conference on Software Maintenance*. IEEE, 1992, pp. 309–317.
- [P33] P. Chung and R. Goodwin, “Representing design history,” in *Artificial Intelligence in Design’94*. Springer, 1994, pp. 735–752.
- [P34] P. B. Kruchten, “The 4+ 1 view model of architecture,” *IEEE software*, vol. 12, no. 6, pp. 42–50, 1995.
- [P35] A. Ran and J. Kuusela, “Design decision trees,” in *Proceedings of the 8th International Workshop on Software Specification and Design*. IEEE Computer Society, 1996, p. 172.
- [P36] P. Kruchten, “The software architect,” in *Software Architecture*. Springer, 1999, pp. 565–583.

- 
- [P37] J. Tyree, “Architectural design decisions session report,” in *5th Working IEEE/IFIP Conference on Software Architecture. WICSA 2005*. IEEE, 2005, pp. 285–286.
- [P38] R. Kazman, J. Asundi, and M. Klein, “Quantifying the costs and benefits of architectural decisions,” in *Proceedings of the 23rd International Conference on Software Engineering. ICSE 2001*. IEEE, 2001, pp. 297–306.
- [P39] M. Moore, R. Kazman, M. Klein, and J. Asundi, “Quantifying the value of architecture design decisions: lessons from the field,” in *Proceedings of the 25th International Conference on Software Engineering*. IEEE Computer Society, 2003, pp. 557–562.
- [P40] P. C. Clements, “An economic model for software architecture decisions,” in *Proceedings of the First international Workshop on the Economics of Software and Computation*. IEEE Computer Society, 2007, p. 1.
- [P41] J. Carriere, R. Kazman, and I. Ozkaya, “A cost-benefit framework for making architectural decisions in a business context,” in *ACM/IEEE 32nd International Conference on Software Engineering, 2010*, vol. 2. IEEE, 2010, pp. 149–157.
- [P42] A. Ran, “Fundamental concepts for practical software architecture,” *ACM SIGSOFT Software Engineering Notes*, vol. 26, no. 5, pp. 328–329, 2001.
- [P43] J. Bosch, “Software architecture: The next step,” in *European Workshop on Software Architecture*. Springer, 2004, pp. 194–199.
- [P44] A. Jansen and J. Bosch, “Evaluation of tool support for architectural evolution,” in *Proceedings of the 19th IEEE International Conference on Automated Software Engineering*. IEEE Computer Society, 2004, pp. 375–378.
- [P45] A. Jansen, J. Bosch, and P. Avgeriou, “Documenting after the fact: Recovering architectural design decisions,” *Journal of Systems and Software*, vol. 81, no. 4, pp. 536–557, 2008.
- [P46] J. S. van der Ven and J. Bosch, “Making the right decision: supporting architects with design decision data,” in *European Conference on Software Architecture*. Springer, 2013, pp. 176–183.
- [P47] A. J. Ko and P. K. Chilana, “Design, discussion, and dissent in open bug reports,” in *Proceedings of the 2011 iConference*. ACM, 2011, pp. 106–113.
- [P48] J. C. Dueñas and R. Capilla, “The decision view of software architecture,” in *European Workshop on Software Architecture*. Springer, 2005, pp. 222–230.
- [P49] A. H. Eden, “Strategic versus tactical design,” in *Proceedings of the 38th Annual Hawaii International Conference on System Sciences. HICSS’05*. IEEE, 2005, pp. 313a–313a.
- [P50] J. Tyree and A. Akerman, “Architecture decisions: Demystifying architecture,” *IEEE software*, vol. 22, no. 2, pp. 19–27, 2005.
- [P51] A. Zalewski and M. Ludzia, “Diagrammatic modeling of architectural decisions,” in *European Conference on Software Architecture*. Springer, 2008, pp. 350–353.

- [P52] Q. Gu, P. Lago, and H. Van Vliet, “A template for soa design decision making in an educational setting,” in *36th EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA 2010)*. IEEE, 2010, pp. 175–182.
- [P53] D. Dermeval, J. Pimentel, C. Silva, J. Castro, E. Santos, G. Guedes, A. Finkelstein *et al.*, “Stream-add-supporting the documentation of architectural design decisions in an architecture derivation process,” in *IEEE 36th Annual Computer Software and Applications Conference (COMPSAC)*. IEEE, 2012, pp. 602–611.
- [P54] V. Clerc, P. Lago, and H. Van Vliet, “The architect’s mindset,” in *International Conference on the Quality of Software Architectures*. Springer, 2007, pp. 231–249.
- [P55] N. Medvidovic and R. N. Taylor, “Software architecture: foundations, theory, and practice,” in *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 2*. ACM, 2010, pp. 471–472.
- [P56] L. Lee and P. Kruchten, “Capturing software architectural design decisions,” in *Canadian Conference on Electrical and Computer Engineering. CCECE 2007*. IEEE, 2007, pp. 686–689.
- [P57] N. B. Harrison, P. Avgeriou, and U. Zdun, “Using patterns to capture architectural decisions,” *IEEE Software*, vol. 24, no. 4, 2007.
- [P58] U. van Heesch, P. Avgeriou, U. Zdun, and N. Harrison, “The supportive effect of patterns in architecture decision recovery—a controlled experiment,” *Science of Computer Programming*, vol. 77, no. 5, 2012.
- [P59] M. T. T. That, S. Sadou, and F. Oquendo, “Using architectural patterns to define architectural decisions,” in *Joint Working IEEE/IFIP Conference on Software Architecture (WICSA) and European Conference on Software Architecture (ECSA)*. IEEE, 2012, pp. 196–200.
- [P60] W. Heijstek, T. Kuhne, and M. R. Chaudron, “Experimental analysis of textual and graphical representations for software architecture design,” in *International Symposium on Empirical Software Engineering and Measurement (ESEM)*. IEEE, 2011, pp. 167–176.
- [P61] A. Calvagna and E. Tramontana, “Delivering dependable reusable components by expressing and enforcing design decisions,” in *IEEE 37th Annual Computer Software and Applications Conference Workshops (COMPSACW)*. IEEE, 2013, pp. 493–498.
- [P62] D. Tofan, M. Galster, P. Avgeriou, and W. Schuitema, “Past and future of software architectural decisions—a systematic mapping study,” *Information and Software Technology*, vol. 56, no. 8, pp. 850–872, 2014.
- [P63] C. Miesbauer and R. Weinreich, “Classification of design decisions—an expert survey in practice,” in *European Conference on Software Architecture*. Springer, 2013, pp. 130–145.
- [P64] A. Jansen and J. Bosch, “Software architecture as a set of architectural design decisions,” in *5th Working IEEE/IFIP Conference on Software Architecture. WICSA 2005*. IEEE,

- 2005, pp. 109–120.
- [P65] A. Akerman and J. Tyree, “Position on ontology-based architecture,” in *5th Working IEEE/IFIP Conference on Software Architecture. WICSA 2005*. IEEE, 2005, pp. 289–290.
- [P66] M. L. Roldán, S. Gonnet, and H. Leone, “A model for capturing and tracing architectural designs,” in *Advanced Software Engineering: Expanding the Frontiers of Software Technology*. Springer, 2006, pp. 16–31.
- [P67] O. Zimmermann, T. Gschwind, J. Küster, F. Leymann, and N. Schuster, “Reusable architectural decision models for enterprise application development,” in *International Conference on the Quality of Software Architectures*. Springer, 2007, pp. 15–32.
- [P68] R. Farenhorst, P. Lago, and H. Van Vliet, “Prerequisites for successful architectural knowledge sharing,” in *Australian Software Engineering Conference (ASWEC’07)*. IEEE, 2007, pp. 27–38.
- [P69] M. Konersmann, Z. Durdik, M. Goedicke, and R. H. Reussner, “Towards architecture-centric evolution of long-living systems (the advert approach),” in *Proceedings of the 9th international ACM Sigsoft conference on Quality of software architectures*. ACM, 2013, pp. 163–168.
- [P70] S. Gerdes, S. Lehnert, and M. Riebisch, “Combining architectural design decisions and legacy system evolution,” in *European Conference on Software Architecture*. Springer, 2014, pp. 50–57.
- [P71] R. C. De Boer, R. Farenhorst, P. Lago, H. Van Vliet, V. Clerc, and A. Jansen, “Architectural knowledge: Getting to the core,” in *International Conference on the Quality of Software Architectures*. Springer, 2007, pp. 197–214.
- [P72] F. Gilson and V. Englebort, “Rationale, decisions and alternatives traceability for architecture design,” in *Proceedings of the 5th European Conference on Software Architecture: Companion Volume*. ACM, 2011, p. 4.
- [P73] B. Orlic, R. Mak, I. David, and J. Lukkien, “Concepts and diagram elements for architectural knowledge management,” in *Proceedings of the 5th European Conference on Software Architecture: Companion Volume*. ACM, 2011, p. 3.
- [P74] Z. Durdik, A. Koziolk, and R. H. Reussner, “How the understanding of the effects of design decisions informs requirements engineering,” in *2nd International Workshop on the Twin Peaks of Requirements and Architecture (TwinPeaks)*. IEEE, 2013, pp. 14–18.
- [P75] L. Zhu and I. Gorton, “Uml profiles for design decisions and non-functional requirements,” in *Proceedings of the Second Workshop on Sharing and Reusing Architectural Knowledge Architecture, Rationale, and Design intent*. IEEE Computer Society, 2007, p. 8.
- [P76] M. Küster, “Architecture-centric modeling of design decisions for validation and traceability,” in *European Conference on Software Architecture*. Springer, 2013, pp. 184–191.
- [P77] Y. Choi, H. Choi, and M. Oh, “An architectural design decision-centric approach to

- architectural evolution,” in *11th International Conference on Advanced Communication Technology. ICACT 2009.*, vol. 1. IEEE, 2009, pp. 417–422.
- [P78] I. Lytra, H. Tran, and U. Zdun, “Constraint-based consistency checking between design decisions and component models for supporting software architecture evolution,” in *16th European Conference on Software Maintenance and Reengineering (CSMR)*. IEEE, 2012, pp. 287–296.
- [P79] M. Soliman and M. Riebisch, “Modeling the interactions between decisions within software architecture knowledge,” in *European Conference on Software Architecture*. Springer, 2014, pp. 33–40.
- [P80] M. Szlenk, A. Zalewski, and S. Kijas, “Modelling architectural decisions under changing requirements,” in *Joint Working IEEE/IFIP Conference on Software Architecture (WICSA) and European Conference on Software Architecture (ECSA)*. IEEE, 2012, pp. 211–214.
- [P81] E. Navarro, C. E. Cuesta, and D. E. Perry, “Weaving a network of architectural knowledge,” in *Joint Working IEEE/IFIP Conference on Software Architecture & European Conference on Software Architecture. WICSA/ECSA 2009*. IEEE, 2009, pp. 241–244.
- [P82] J. P. Ros and R. S. Sangwan, “A method for evidence-based architecture discovery,” in *9th Working IEEE/IFIP Conference on Software Architecture (WICSA)*. IEEE, 2011, pp. 342–345.
- [P83] A. Sawada, M. Noro, H.-M. Chang, Y. Hachisu, and A. Yoshida, “A design map for recording precise architecture decisions,” in *18th Asia-Pacific Software Engineering Conference*. IEEE, 2011, pp. 298–305.
- [P84] D. Dermeval, J. Castro, C. Silva, J. Pimentel, I. I. Bittencourt, P. Brito, E. Elias, T. Tenório, and A. Pedro, “On the use of metamodeling for relating requirements and architectural design decisions,” in *Proceedings of the 28th Annual ACM Symposium on Applied Computing*. ACM, 2013, pp. 1278–1283.
- [P85] M. Mirakhorli and J. Cleland-Huang, “Transforming trace information in architectural documents into re-usable and effective traceability links,” in *Proceedings of the 6th International Workshop on SHaring and Reusing Architectural Knowledge*. ACM, 2011, pp. 45–52.
- [P86] M. Mirakhorli, “Tracing architecturally significant requirements: a decision-centric approach,” in *Proceedings of the 33rd International Conference on Software Engineering*. ACM, 2011, pp. 1126–1127.
- [P87] R. Capilla and M. A. Babar, “On the role of architectural design decisions in software product line engineering,” in *European Conference on Software Architecture*. Springer, 2008, pp. 241–255.
- [P88] P. G. Avery and R. D. Hawkins, “Software design decision vulnerability analysis,” in *9th IET International Conference on System Safety and Cyber Security*. IET, 2014.
- [P89] A. Tang and H. Van Vliet, “Modeling constraints improves software architecture design

- reasoning,” in *Joint Working IEEE/IFIP Conference on Software Architecture & European Conference on Software Architecture. WICSA/ECSA 2009*. IEEE, 2009, pp. 253–256.
- [P90] A. Tang and H. Van Vliet, “Software architecture design reasoning,” in *Software Architecture Knowledge Management*. Springer, 2009, pp. 155–174.
- [P91] M. Che and D. E. Perry, “Scenario-based architectural design decisions documentation and evolution,” in *18th IEEE International Conference and Workshops on Engineering of Computer Based Systems (ECBS)*. IEEE, 2011, pp. 216–225.
- [P92] M. Che, “An approach to documenting and evolving architectural design decisions,” in *Proceedings of the 2013 International Conference on Software Engineering*. IEEE Press, 2013, pp. 1373–1376.
- [P93] O. Zimmermann, “Architectural decisions as reusable design assets,” *IEEE software*, vol. 28, no. 1, pp. 64–69, 2011.
- [P94] C. Carrillo, R. Capilla, O. Zimmermann, and U. Zdun, “Guidelines and metrics for configurable and sustainable architectural knowledge modelling,” in *Proceedings of the 2015 European Conference on Software Architecture Workshops*. ACM, 2015, p. 63.
- [P95] J. Carlson, E. Papatheocharous, and K. Petersen, “A context model for architectural decision support,” in *1st International Workshop on Decision Making in Software Architecture (MARCH)*. IEEE, 2016, pp. 9–15.
- [P96] I. Lytra, P. Gaubatz, and U. Zdun, “Two controlled experiments on model-based architectural decision making,” *Information and Software Technology*, vol. 63, pp. 58–75, 2015.
- [P97] S. Stevanetic, K. Plakidas, T. B. Ionescu, F. Li, D. Schall, and U. Zdun, “Tool support for the architectural design decisions in software ecosystems,” in *Proceedings of the 2015 European Conference on Software Architecture Workshops*. ACM, 2015, p. 45.
- [P98] I. Lytra, G. Engelbrecht, D. Schall, and U. Zdun, “Reusable architectural decision models for quality-driven decision support: A case study from a smart cities software ecosystem,” in *Proceedings of the Third International Workshop on Software Engineering for Systems-of-Systems*. IEEE Press, 2015, pp. 37–43.
- [P99] M. Soliman, M. Riebisch, and U. Zdun, “Enriching architecture knowledge with technology design decisions,” in *12th Working IEEE/IFIP Conference on Software Architecture (WICSA)*. IEEE, 2015, pp. 135–144.
- [P100] C. C. Venters, R. Capilla, S. Betz, B. Penzenstadler, T. Crick, S. Crouch, E. Y. Nakagawa, C. Becker, and C. Carrillo, “Software sustainability: Research and practice from a software architecture viewpoint,” *Journal of Systems and Software*, 2017.
- [P101] U. Zdun, R. Capilla, H. Tran, and O. Zimmermann, “Sustainable architectural design decisions,” *IEEE Software*, vol. 30, no. 6, pp. 46–53, 2013.
- [P102] M. Nowak and C. Pautasso, “Goals, questions and metrics for architectural decision

- models,” in *Proceedings of the 6th International Workshop on SHaring and Reusing Architectural Knowledge*. ACM, 2011, pp. 21–28.
- [P103] M. Shahin, P. Liang, and M. R. Khayyambashi, “Architectural design decision: Existing models and tools,” in *Joint Working IEEE/IFIP Conference on Software Architecture & European Conference on Software Architecture. WICSA/ECSA 2009*. IEEE, 2009, pp. 293–296.
- [P104] D. Ameller, C. Ayala, J. Cabot, and X. Franch, “Non-functional requirements in architectural decision making,” *IEEE Software*, vol. 30, no. 2, pp. 61–67, 2013.
- [P105] J. E. Robbins and D. F. Redmiles, “Cognitive support, uml adherence, and xmi interchange in argo/uml,” *Information and Software Technology*, vol. 42, no. 2, pp. 79–89, 2000.
- [P106] R. Capilla, F. Nava, S. Pérez, and J. C. Dueñas, “A web-based tool for managing architectural design decisions,” *ACM SIGSOFT software engineering notes*, vol. 31, no. 5, p. 4, 2006.
- [P107] R. Capilla, F. Nava, J. Montes, and C. Carrillo, “Addss: architecture design decision support system tool,” in *Proceedings of the 2008 23rd IEEE/ACM International Conference on Automated Software Engineering*. IEEE Computer Society, 2008, pp. 487–488.
- [P108] R. Capilla and F. Nava, “Extending software architecting processes with decision-making activities,” in *Balancing Agility and Formalism in Software Engineering*. Springer, 2008, pp. 182–195.
- [P109] R. Capilla, F. Nava, and J. C. Duenas, “Modeling and documenting the evolution of architectural design decisions,” in *Proceedings of the Second Workshop on SHaring and Reusing architectural Knowledge Architecture, Rationale, and Design Intent*. IEEE Computer Society, 2007, p. 9.
- [P110] R. Capilla, F. Nava, and A. Tang, “Attributes for characterizing the evolution of architectural design decisions,” in *3rd International IEEE Workshop on Software Evolvability*. IEEE, 2007, pp. 15–22.
- [P111] F. Nava, R. Capilla, and J. C. Dueñas, “Processes for creating and exploiting architectural design decisions with tool support,” in *European Conference on Software Architecture*. Springer, 2007, pp. 321–324.
- [P112] R. Capilla, F. Nava, and C. Carrillo, “Effort estimation in capturing architectural knowledge,” in *Proceedings of the 2008 23rd IEEE/ACM International Conference on Automated Software Engineering*. IEEE Computer Society, 2008, pp. 208–217.
- [P113] R. Capilla, “Embedded design rationale in software architecture,” in *Joint Working IEEE/IFIP Conference on Software Architecture & European Conference on Software Architecture. WICSA/ECSA 2009*. IEEE, 2009, pp. 305–308.
- [P114] L. Lee and P. Kruchten, “A tool to visualize architectural design decisions,” in *International Conference on the Quality of Software Architectures*. Springer, 2008, pp. 43–54.



- [P115] L. Lee and P. Kruchten, “Visualizing software architectural design decisions,” in *European Conference on Software Architecture*. Springer, 2008, pp. 359–362.
- [P116] L. Lee and P. Kruchten, “Customizing the capture of software architectural design decisions,” in *Canadian Conference on Electrical and Computer Engineering. CCECE 2008*. IEEE, 2008, pp. 000 693–000 698.
- [P117] R. C. De Boer, P. Lago, A. Telea, and H. Van Vliet, “Ontology-driven visualization of architectural design decisions,” in *Joint Working IEEE/IFIP Conference on Software Architecture & European Conference on Software Architecture. WICSA/ECSA 2009*. IEEE, 2009, pp. 51–60.
- [P118] M. Shahin, P. Liang, and M. R. Khayyambashi, “Rationale visualization of software architectural design decision using compendium,” in *Proceedings of the 2010 ACM Symposium on Applied Computing*. ACM, 2010, pp. 2367–2368.
- [P119] M. Shahin and P. Liang and M. R. Khayyambashi, “Improving understandability of architecture design through visualization of architectural design decision,” in *Proceedings of the 2010 ICSE Workshop on Sharing and Reusing Architectural Knowledge*. ACM, 2010, pp. 88–95.
- [P120] M. Shahin, P. Liang, and Z. Li, “Architectural design decision visualization for architecture design: preliminary results of a controlled experiment,” in *Proceedings of the 5th European Conference on Software Architecture: Companion Volume*. ACM, 2011, p. 2.
- [P121] M. Shahin and P. Liang and Z. Li, “Do architectural design decisions improve the understanding of software architecture? two controlled experiments,” in *Proceedings of the 22nd International Conference on Program Comprehension*. ACM, 2014, pp. 3–13.
- [P122] P. Konemann, “Integrating decision management with uml modeling concepts and tools,” in *Joint Working IEEE/IFIP Conference on Software Architecture & European Conference on Software Architecture. WICSA/ECSA 2009*. IEEE, 2009, pp. 297–300.
- [P123] L. Chen and M. A. Babar, “Supporting customizable architectural design decision management,” in *17th IEEE International Conference and Workshops on Engineering of Computer Based Systems (ECBS)*. IEEE, 2010, pp. 232–240.
- [P124] L. Chen, M. A. Babar, and H. Liang, “Model-centered customizable architectural design decisions management,” in *21st Australian Software Engineering Conference*. IEEE, 2010, pp. 23–32.
- [P125] G. Buchgeher and R. Weinreich, “Automatic tracing of decisions to architecture and implementation,” in *9th Working IEEE-IFIP Conference on Software Architecture*. IEEE, 2011, pp. 46–55.
- [P126] H. Astudillo, G. Valdés, and C. Becerra, “Empirical measurement of automated recovery of design decisions and structure,” in *Proceedings of the 2012 Andean Region International Conference*. IEEE Computer Society, 2012, pp. 105–108.
- [P127] K. Nakakoji, Y. Yamamoto, N. Matsubara, and Y. Shirai, “Toward unweaving streams of thought for reflection in professional software design,” *IEEE Software*, vol. 29, no. 1,

- pp. 34–38, 2012.
- [P128] J. M. E. van der Werf, R. de Feijter, F. Bex, and S. Brinkkemper, “Facilitating collaborative decision making with the software architecture video wall,” in *International Conference on Software Architecture Workshops (ICSAW)*. IEEE, 2017, pp. 137–140.
- [P129] G. Pedraza-Garcia, H. Astudillo, and D. Correal, “Analysis of design meetings for understanding software architecture decisions,” in *XL Latin American Computing Conference (CLEI)*. IEEE, 2014, pp. 1–10.
- [P130] G. Pedraza-García, H. Astudillo, and D. Correal, “Dvia: Understanding how software architects make decisions in design meetings,” in *Proceedings of the 2015 European Conference on Software Architecture Workshops*. ACM, 2015, p. 51.
- [P131] G. Pedraza-Garcia, H. Astudillo, and D. Correal, “An approach for software knowledge sharing based on architectural decisions,” in *2016 XLII Latin American Computing Conference (CLEI)*. IEEE, 2016, pp. 1–10.
- [P132] C. Dhaya and G. Zayaraz, “Development of multiple architectural designs using aduak,” in *International Conference on Communications and Signal Processing (ICCSPP)*. IEEE, 2012, pp. 93–97.
- [P133] D. Ameller, O. Collell, and X. Franch, “Architech: Tool support for nfr-guided architectural decision-making,” in *20th IEEE International Requirements Engineering Conference (RE)*. IEEE, 2012, pp. 315–316.
- [P134] A. Gopalakrishnan and A. C. Biswal, “Quiver an intelligent decision support system for software architecture and design,” in *International Conference On Smart Technologies For Smart Nation (SmartTechCon)*. IEEE, 2017, pp. 1286–1291.
- [P135] Z. Durdik and R. Reussner, “Position paper: approach for architectural design and modelling with documented design decisions (admd3),” in *Proceedings of the 8th international ACM SIGSOFT conference on Quality of Software Architectures*. ACM, 2012, pp. 49–54.
- [P136] I. Lytra, H. Tran, and U. Zdun, “Supporting consistency between architectural design decisions and component models through reusable architectural knowledge transformations,” in *European Conference on Software Architecture*. Springer, 2013, pp. 224–239.
- [P137] D. Tofan and M. Galster, “Capturing and making architectural decisions: an open source online tool,” in *Proceedings of the 2014 European Conference on Software Architecture Workshops*. ACM, 2014, p. 33.
- [P138] D. Tofan, M. Galster, and P. Avgeriou, “Capturing tacit architectural knowledge using the repertory grid technique (nier track),” in *Proceedings of the 33rd International Conference on Software Engineering*. ACM, 2011, pp. 916–919.
- [P139] C. Manteuffel, D. Tofan, H. Koziol, T. Goldschmidt, and P. Avgeriou, “Industrial implementation of a documentation framework for architectural decisions,” in *IEEE/IFIP Conference on Software Architecture (WICSA)*. IEEE, 2014, pp. 225–234.
- [P140] T.-M. Hesse, A. Kuehlwein, and T. Roehm, “Decdoc: A tool for documenting design

- decisions collaboratively and incrementally,” in *1st International Workshop on Decision Making in Software ARCHitecture (MARCH)*. IEEE, 2016, pp. 30–37.
- [P141] Z. Alexeeva, D. Perez-Palacin, and R. Mirandola, “Design decision documentation: A literature overview,” in *European Conference on Software Architecture*. Springer, 2016, pp. 84–101.
- [P142] J. Burge and D. C. Brown, “Reasoning with design rationale,” in *Artificial Intelligence in Design’00*. Springer, 2000, pp. 611–629.
- [P143] J. E. Burge and D. C. Brown, “An integrated approach for software design checking using design rationale,” in *Design Computing and Cognition’04*. Springer, 2004, pp. 557–575.
- [P144] J. E. Burge and J. D. Kiper, “Capturing decisions and rationale from collaborative design,” in *Design computing and cognition’08*. Springer, 2008, pp. 221–239.
- [P145] W. Wang and J. E. Burge, “Using rationale to support pattern-based architectural design,” in *Proceedings of the 2010 ICSE Workshop on Sharing and Reusing Architectural Knowledge*. ACM, 2010, pp. 1–8.
- [P146] I. Douglas, “Capturing and managing decision making rationale,” in *IRI-2005 IEEE International Conference on Information Reuse and Integration*. IEEE, 2005, pp. 172–176.
- [P147] D. Falessi, R. Capilla, and G. Cantone, “A value-based approach for documenting design decisions rationale: a replicated experiment,” in *Proceedings of the 3rd international workshop on Sharing and reusing architectural knowledge*. ACM, 2008, pp. 63–70.
- [P148] D. Falessi, L. C. Briand, G. Cantone, R. Capilla, and P. Kruchten, “The value of design rationale information,” *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 22, no. 3, p. 21, 2013.
- [P149] S. Sundaravadivelu, A. Vaidyanathan, and S. Ramaswamy, “Knowledge reuse of software architecture design decisions and rationale within the enterprise,” in *International Conference on Issues and Challenges in Intelligent Computing Techniques (ICICT)*. IEEE, 2014, pp. 253–261.
- [P150] N. Chanda and X. F. Liu, “Intelligent analysis of software architecture rationale for collaborative software design,” in *International Conference on Collaboration Technologies and Systems (CTS)*. IEEE, 2015, pp. 287–294.
- [P151] A. Tang, M. A. Babar, I. Gorton, and J. Han, “A survey of the use and documentation of architecture design rationale,” in *5th Working IEEE/IFIP Conference on Software Architecture. WICSA 2005*. IEEE, 2005, pp. 89–98.
- [P152] D. Falessi, G. Cantone, R. Kazman, and P. Kruchten, “Decision-making techniques for software architecture design: A comparative survey,” *ACM Computing Surveys (CSUR)*, vol. 43, no. 4, p. 33, 2011.
- [P153] R. Rauscher, “A design assistant for scheduling of design decisions,” in *Proceedings of EURO MICRO 96. 22nd Euromicro Conference. Beyond 2000: Hardware and Software*

- Design Strategies*. IEEE, 1996, p. 0088.
- [P154] L. Borrman and F. N. Paulisch, “Software architecture at siemens: The challenges, our approaches, and some open issues,” in *Software Architecture*. Springer, 1999, pp. 529–543.
- [P155] C. Zannier and F. Maurer, “A qualitative empirical evaluation of design decisions,” in *ACM SIGSOFT Software Engineering Notes*, vol. 30, no. 4. ACM, 2005, pp. 1–7.
- [P156] C. Zannier and F. Maurer, “Foundations of agile decision making from agile mentors and developers,” in *International Conference on Extreme Programming and Agile Processes in Software Engineering*. Springer, 2006, pp. 11–20.
- [P157] C. Zannier and F. Maurer, “Comparing decision making in agile and non-agile software organizations,” in *International Conference on Extreme Programming and Agile Processes in Software Engineering*. Springer, 2007, pp. 1–8.
- [P158] C. Zannier, M. Chiasson, and F. Maurer, “A model of design decision making based on empirical results of interviews with software designers,” *Information and Software Technology*, vol. 49, no. 6, pp. 637–653, 2007.
- [P159] C. Zannier and F. Maurer, “Social factors relevant to capturing design decisions,” in *Proceedings of the 2nd Workshop on SHaring and Reusing architectural Knowledge Architecture, Rationale, and Design Intent*. IEEE Computer Society, 2007, p. 1.
- [P160] J. Hutchinson and G. Kotonya, “A review of negotiation techniques in component based software engineering,” in *32nd EUROMICRO Conference on Software Engineering and Advanced Applications. SEAA’06*. IEEE, 2006, pp. 152–159.
- [P161] M. Anvaari, R. Conradi, and L. Jaccheri, “Architectural decision-making in enterprises: preliminary findings from an exploratory study in norwegian electricity industry,” in *European Conference on Software Architecture*. Springer, 2013, pp. 162–175.
- [P162] S. Dasanayake, J. Markkula, S. Aaramaa, and M. Oivo, “Software architecture decision-making practices and challenges: an industrial case study,” in *24th Australian Software Engineering Conference (ASWEC)*. IEEE, 2015, pp. 88–97.
- [P163] A. Tang, M. H. Tran, J. Han, and H. Van Vliet, “Design reasoning improves software design quality,” in *International Conference on the Quality of Software Architectures*. Springer, 2008, pp. 28–42.
- [P164] A. Tang, A. Aleti, J. Burge, and H. van Vliet, “What makes software design effective?” *Design Studies*, vol. 31, no. 6, pp. 614–640, 2010.
- [P165] A. Tang and H. van Vliet, “Software designers satisfice,” in *European Conference on Software Architecture*. Springer, 2015, pp. 105–120.
- [P166] C. Yang, P. Liang, P. Avgeriou, U. Eliasson, R. Heldal, and P. Pelliccione, “Architectural assumptions and their management in industry—an exploratory study,” in *European Conference on Software Architecture*. Springer, 2017, pp. 191–207.
- [P167] S. T. Hassard, A. Blandford, and A. L. Cox, “Analogies in design decision-making,” in

- Proceedings of the 23rd British HCI group annual conference on people and computers: celebrating people and technology.* British Computer Society, 2009, pp. 140–148.
- [P168] H. Christiaans and R. A. Almendra, “Accessing decision-making in software design,” *Design Studies*, vol. 31, no. 6, pp. 641–662, 2010.
- [P169] E. Papatheocharous, K. Petersen, A. Cicchetti, S. Sentilles, S. M. A. Shah, and T. Gorschek, “Decision support for choosing architectural assets in the development of software-intensive systems: The grade taxonomy,” in *Proceedings of the 2015 European Conference on Software Architecture Workshops*. ACM, 2015, p. 48.
- [P170] C. Wohlin, K. Wnuk, D. Smite, U. Franke, D. Badampudi, and A. Cicchetti, “Supporting strategic decision-making for selection of software assets,” in *International Conference of Software Business*. Springer, 2016, pp. 1–15.
- [P171] A. Cicchetti, M. Borg, S. Sentilles, K. Wnuk, J. Carlson, and E. Papatheocharous, “Towards software assets origin selection supported by a knowledge repository,” in *1st International Workshop on Decision Making in Software ARCHitecture (MARCH)*. IEEE, 2016, pp. 22–29.
- [P172] B. Xu, Z. Huang, and O. Wei, “Making architectural decisions based on requirements: Analysis and combination of risk-based and quality attribute-based methods,” in *7th International Conference on Ubiquitous Intelligence & Computing and 7th International Conference on Autonomic & Trusted Computing (UIC/ATC)*. IEEE, 2010, pp. 392–397.
- [P173] G. Márquez and H. Astudillo, “Selecting components assemblies from non-functional requirements through tactics and scenarios,” in *35th International Conference of the Chilean Computer Science Society (SCCC)*. IEEE, 2016, pp. 1–11.
- [P174] S. Blair, R. Watt, and T. Cull, “Responsibility-driven architecture,” *IEEE software*, vol. 27, no. 2, 2010.
- [P175] X. Cui, Y. Sun, S. Xiao, and H. Mei, “Architecture design for the large-scale software-intensive systems: A decision-oriented approach and the experience,” in *14th IEEE International Conference on Engineering of Complex Computer Systems*. IEEE, 2009, pp. 30–39.
- [P176] I. Lytra, S. Sobernig, and U. Zdun, “Architectural decision making for service-based platform integration: A qualitative multi-method study,” in *Joint Working IEEE/IFIP Conference on Software Architecture (WICSA) and European Conference on Software Architecture (ECSA)*. IEEE, 2012, pp. 111–120.
- [P177] D. Tofan, M. Galster, and P. Avgeriou, “Difficulty of architectural decisions—a survey with professional architects,” in *European Conference on Software Architecture*. Springer, 2013, pp. 192–199.
- [P178] G. Pedraza-Garcia, H. Astudillo, and D. Correal, “Modeling software architecture process with a decision-making approach,” in *33rd International Conference of the Chilean Computer Science Society (SCCC)*. IEEE, 2014, pp. 1–6.
- [P179] A. Dragomir, H. Lichter, and T. Budau, “Systematic architectural decision manage-

- ment, a process-based approach,” in *IEEE/IFIP Conference on Software Architecture (WICSA)*. IEEE, 2014, pp. 255–258.
- [P180] R. J. Wirfs-Brock, “Giving design advice,” *IEEE Software*, vol. 24, no. 4, 2007.
- [P181] A. Zalewski, K. Borowa, and A. Ratkowski, “On cognitive biases in architecture decision making,” in *European Conference on Software Architecture*. Springer, 2017, pp. 123–137.
- [P182] H. van Vliet and A. Tang, “Decision making in software architecture,” *Journal of Systems and Software*, vol. 117, pp. 638–644, 2016.
- [P183] A. Tang, “Software designers, are you biased?” in *Proceedings of the 6th International Workshop on SHaring and Reusing Architectural Knowledge*. ACM, 2011, pp. 1–8.
- [P184] I. Groher and R. Weinreich, “A study on architectural decision-making in context,” in *12th Working IEEE/IFIP Conference on Software Architecture (WICSA)*. IEEE, 2015, pp. 11–20.
- [P185] R. Weinreich, I. Groher, and C. Miesbauer, “An expert survey on kinds, influence factors and documentation of design decisions in practice,” *Future Generation Computer Systems*, vol. 47, pp. 145–160, 2015.
- [P186] N. B. Harrison, E. Gubler, and D. Skinner, “Architectural decision-making in open-source systems—preliminary observations,” in *1st International Workshop on Decision Making in Software ARCHitecture (MARCH)*. IEEE, 2016, pp. 16–21.
- [P187] C. Schriek, J. M. E. van der Werf, A. Tang, and F. Bex, “Software architecture design reasoning: A card game to help novice designers,” in *European Conference on Software Architecture*. Springer, 2016, pp. 22–38.
- [P188] M. Razavian, A. Tang, R. Capilla, and P. Lago, “In two minds: how reflections influence software design thinking,” *Journal of Software: Evolution and Process*, vol. 28, no. 6, pp. 394–426, 2016.
- [P189] J. S. van der Ven and J. Bosch, “Busting software architecture beliefs: A survey on success factors in architecture decision making,” in *42th Euromicro Conference on Software Engineering and Advanced Applications (SEAA), 2016*. IEEE, 2016, pp. 42–49.
- [P190] P. Ralph and E. Tempero, “Characteristics of decision-making during coding,” in *Proceedings of the 20th International Conference on Evaluation and Assessment in Software Engineering*. ACM, 2016, p. 34.
- [P191] M. Razavian, A. Tang, R. Capilla, and P. Lago, “Reflective approach for software design decision making,” in *Qualitative Reasoning about Software Architectures (QRASA)*. IEEE, 2016, pp. 19–26.
- [P192] A. Tang, M. Razavian, B. Paech, and T.-M. Hesse, “Human aspects in software architecture decision making: a literature review,” in *IEEE International Conference on Software Architecture (ICSA)*. IEEE, 2017, pp. 107–116.
- [P193] J. W. Cangussu, K. C. Cooper, and E. W. Wong, “Multi criteria selection of components using the analytic hierarchy process,” in *International Symposium on Component-Based*

- Software Engineering*. Springer, 2006, pp. 67–81.
- [P194] N. Ernst, J. Klein, G. Mathew, and T. Menzies, “Using stakeholder preferences to make better architecture decisions,” in *IEEE International Conference on Software Architecture Workshops (ICSAW)*. IEEE, 2017, pp. 133–136.
- [P195] A. Harchenko, I. Bodnarchuk, I. Halay, and V. Yatsyshyn, “The tool for design of software systems architecture,” in *12th International Conference on the Experience of Designing and Application of CAD Systems in Microelectronics (CADSM)*. IEEE, 2013, pp. 138–139.
- [P196] A. Harchenko, I. Bodnarchuk, and I. Halay, “Decision support system of software architect,” in *IEEE 7th International Conference on Intelligent Data Acquisition and Advanced Computing Systems (IDAACS)*, vol. 1. IEEE, 2013, pp. 265–269.
- [P197] S. Orlov and A. Vishnyakov, “Decision making for the software architecture structure based on the criteria importance theory,” *Procedia Computer Science*, vol. 104, pp. 27–34, 2017.
- [P198] A. Egyed and D. S. Wile, “Support for managing design-time decisions,” *IEEE Transactions on Software Engineering*, no. 5, pp. 299–314, 2006.
- [P199] T. Al-Naeem, F. T. Dabous, F. A. Rabhi, and B. Benatallah, “Formulating the architectural design of enterprise applications as a search problem,” in *Australian Software Engineering Conference*. IEEE, 2005, pp. 282–291.
- [P200] M. Makki, E. Bagheri, and A. A. Ghorbani, “Automating architecture trade-off decision making through a complex multi-attribute decision process,” in *European Conference on Software Architecture*. Springer, 2008, pp. 264–272.
- [P201] M. Riebisch and S. Wohlfarth, “Introducing impact analysis for architectural decisions,” in *14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems. ECBS’07*. IEEE, 2007, pp. 381–392.
- [P202] F. H. Jabali, S. M. Sharafi, and K. Zamanifar, “A quantitative algorithm to select software architecture by tradeoff between quality attributes,” *Procedia Computer Science*, vol. 3, pp. 1480–1484, 2011.
- [P203] L. Grunske, “Identifying good architectural design alternatives with multi-objective optimization strategies,” in *Proceedings of the 28th International Conference on Software Engineering*. ACM, 2006, pp. 849–852.
- [P204] S. Moaven, J. Habibi, H. Ahmadi, and A. Kamandi, “A fuzzy model for solving architecture styles selection multi-criteria problem,” in *2nd UKSIM European Symposium on Computer Modeling and Simulation, 2008. EMS’08*. IEEE, 2008, pp. 388–393.
- [P205] S. Moaven and J. Habibi and H. Ahmadi and A. Kamandi, “A decision support system for software architecture-style selection,” in *6th International Conference on Software Engineering Research, Management and Applications, 2008. SERA’08*. IEEE, 2008, pp. 213–220.

- [P206] G. Zayaraz, S. Vijayalakshmi, and V. Vijayalakshmi, "Evaluation of software architectures using multicriteria fuzzy decision making technique," in *International Conference on Intelligent Agent & Multi-Agent Systems. IAMA 2009*. IEEE, 2009, pp. 1–5.
- [P207] N. Esfahani, S. Malek, and K. Razavi, "Guidearch: guiding the exploration of architectural solution space under uncertainty," in *Proceedings of the 2013 International Conference on Software Engineering*. IEEE Press, 2013, pp. 43–52.
- [P208] S. Gerdes, M. Soliman, and M. Riebisch, "Decision buddy: tool support for constraint-based design decisions during system evolution," in *Proceedings of the 1st International Workshop on Future of Software Architecture Design Assistants*. ACM, 2015, pp. 13–18.
- [P209] S. A. Busari, "Towards search-based modelling and analysis of requirements and architecture decisions," in *32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2017, pp. 1026–1029.
- [P210] N. Upadhyay, "Sdmf: Systematic decision-making framework for evaluation of software architecture," *Procedia computer science*, vol. 91, pp. 599–608, 2016.
- [P211] M. A. Al Imran, S. P. Lee, and M. M. Ahsan, "Quality driven architectural solutions selection approach through measuring impact factors," in *International Conference on Electrical Engineering and Computer Science (ICECOS)*. IEEE, 2017, pp. 131–136.
- [P212] J. A. Díaz-Pace and M. R. Campo, "Exploring alternative software architecture designs: a planning perspective," *IEEE Intelligent Systems*, vol. 23, no. 5, 2008.
- [P213] M. Scheerer, A. Busch, and A. Koziolok, "Automatic evaluation of complex design decisions in component-based software architectures," in *Proceedings of the 15th ACM-IEEE International Conference on Formal Methods and Models for System Design*. ACM, 2017, pp. 67–76.
- [P214] G. Sapienza, G. Dodig-Crnkovic, and I. Crnkovic, "Inclusion of ethical aspects in multicriteria decision analysis," in *1st International Workshop on Decision Making in Software ARCHitecture (MARCH)*. IEEE, 2016, pp. 1–8.
- [P215] A. Shahbazian, Y. K. Lee, Y. Brun, and N. Medvidovic, "Making well-informed software design decisions," in *Proceedings of the 40th International Conference on Software Engineering: Companion Proceedings*. ACM, 2018, pp. 262–263.
- [P216] A. Alali and J. Sillito, "Motivations for collaboration in software design decision making," in *6th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE)*. IEEE, 2013, pp. 129–132.
- [P217] S. Dasanayake, J. Markkula, S. Aaramaa, and M. Oivo, "An empirical study on collaborative architecture decision making in software teams," in *European Conference on Software Architecture*. Springer, 2016, pp. 238–246.
- [P218] J. Chai and J. N. Liu, "An ontology-driven framework for supporting complex decision process," in *World Automation Congress (WAC)*. IEEE, 2010, pp. 1–6.
- [P219] M. Nowak and C. Pautasso, "Team situational awareness and architectural decision mak-



- ing with the software architecture warehouse,” in *European Conference on Software Architecture*. Springer, 2013, pp. 146–161.
- [P220] D. Tofan, M. Galster, I. Lytra, P. Avgeriou, U. Zdun, M.-A. Fouche, R. De Boer, and F. Solms, “Empirical evaluation of a process to increase consensus in group architectural decision making,” *Information and Software Technology*, vol. 72, pp. 31–47, 2016.
- [P221] S. V. F. Lopes and P. T. A. Junior, “Architectural design group decision-making in agile projects,” in *IEEE International Conference on Software Architecture Workshops (ICSAW)*. IEEE, 2017, pp. 210–215.
- [P222] V. S. Rekhav and H. Muccini, “A study on group decision-making in software architecture,” in *2014 IEEE/IFIP Conference on Software Architecture (WICSA)*. IEEE, 2014, pp. 185–194.
- [P223] S. Rekha and H. Muccini, “Suitability of software architecture decision making methods for group decisions,” in *European Conference on Software Architecture*. Springer, 2014, pp. 17–32.
- [P224] I. Malavolta, H. Muccini, and S. Rekha, “Enhancing architecture design decisions evolution with group decision making principles,” in *International Workshop on Software Engineering for Resilient Systems*. Springer, 2014, pp. 9–23.
- [P225] H. Muccini, D. A. Tamburri, and V. S. Rekha, “On the social dimensions of architectural decisions,” in *European Conference on Software Architecture*. Springer, 2015, pp. 137–145.
- [P226] S. Rekha and H. Muccini, “Group decision-making in software architecture: A study on industrial practices,” *Information and Software Technology*, 2018.
- [P227] R. Capilla, A. Jansen, A. Tang, P. Avgeriou, and M. A. Babar, “10 years of software architecture knowledge management: Practice and future,” *Journal of Systems and Software*, vol. 116, pp. 191–205, 2016.



---

## Abbreviations

---

- ADD** Architectural Design Decision
- ADDs** Architectural Design Decisions
- ADeX** Amelie - Decision eXplorer
- ADM** Architectural Decision Making
- ADR** Architectural Design Rationale
- AEs** Architectural Elements
- AK** Architectural Knowledge
- AKM** Architectural Knowledge Management
- ASRs** Architecturally Significant Requirements
- BRM** Bounded Rational Model
- DMMs** Decision Making Models
- EA** Enterprise Architecture
- ETL** Extract, Transform, and Load

**GDM** Group Decision Making

**IBIS** Issue-Based Information Systems

**IMS** Issue Management System

**IMs** Issue Management Systems

**KB** Knowledge Base

**MCDM** Multiple-Criteria Decision-Making

**ML** Machine Learning

**NDM** Naturalistic Decision Making

**NFRs** Non-Functional Requirements

**NLP** Natural Language Processing

**OSS** Open Source Software

**QOC** Question, Option, Criteria

**RDM** Rationalistic Decision Making

**REM** Rational Economic Model

**RPDM** Recognition-Primed Decision Model

**SA** Software Architecture

**SDLC** Software Development Lifecycle

**SE** Software Engineering

**UI** User Interface

**UIs** User Interfaces

**UML** Unified Modeling Language