2015 12th Working IEEE/IFIP Conference on Software Architecture

Enriching Architecture Knowledge with Technology Design Decisions

Mohamed Soliman*, Matthias Riebisch* and Uwe Zdun†
*Department of Informatics, University of Hamburg, Germany
Email: {soliman, riebisch}@informatik.uni-hamburg.de
†Software Architecture Research Group, University of Vienna, Austria
Email: uwe.zdun@univie.ac.at

***Abstract*—Decision-making is at the core of software architecture design. However, in order for the architect to take the right design decisions, assistance is required for exploring the architectural knowledge, which encompasses the various architectural solutions, their relationships and distinctions. In the past decades, the number of available technology options has increased significantly, while existing architecture knowledge approaches support technology decisions by representing relations between the different technology solutions, as well as design problems. However, they do not differentiate the candidate technologies according to their offered qualities and drawbacks. Our main goal in this exploratory study is to understand how technology solutions are being considered by the architects during the design process, and how can we enhance existing architecture knowledge concepts to support technology decision making. Our contribution in this paper is differentiating the different technology solutions' features based on a set of architecturally significant aspects, to facilitate considering technologies during the architecture design decisions. In addition, we proposed an extension for existing architecture knowledge models, which characterize the technology design decisions, and their reasoning. We evaluated our results through real examples from practitioners. Moreover, we conducted interviews with experts to validate our proposed concepts.**

I. Introduction

Taking the right design decisions for crucial design issues is a core aspect of software architecture. These decisions are distinguished by their associated risks, which originate from the various unknowns that surround the decision maker during the design process. One of the reasons for such uncertainty is due to the amount and diversity of the architectural solutions, and their different capabilities to satisfy the system's *Architecture Significant Requirements (ASRs)* [1]. As a consequence, it is arduous to select accurately the appropriate solutions for the different architecture design problems.

In order to assist the architect in exploring the design space and selecting the right combination of architectural solutions, approaches have been proposed to model and manage the architecture knowledge. Recent approaches are based on modeling the possible *Architectural Design Decisions (ADDs)* [2]. Design issues, architectural solutions, and the evaluations for their different combinations are the main building blocks of this architecture knowledge, which is characterized by its continuous evolution.

Architectural solutions could be classified into *Conceptual Solutions* and *Technology Solutions* [3]. The former are abstract solution's notions for design problems, which could be implemented differently in different contexts. This type include patterns (e.g. architectural patterns [4] and design patterns [5]), as well as design tactics [6]. On the other hand, technology solutions are concrete solutions, which assist the architect to develop and implement the system. This include frameworks, programming languages, standards and libraries. Both types of solutions are interrelated. In addition, both types impact the system functionality and quality differently.

Within the past decades, the selection of the right technology product have turned out to be gradually complicated, due to the significant increase in the production of software products by technology vendors, as well as the open source community [7].

A recent survey [8] on the types of architectural design decisions and their documentation shows that around 25% of a system decisions are executive decisions, and most of them are technology decisions. Moreover, technology decisions as well as structural decisions have been observed as the mostly documented design decisions among other categories of decisions. The survey participants indicated that technology decisions are taken early in the design process, and it is quite hard to change them during the system implementation. These results indicate and affirm the importance of technology decisions for the software architect.

Despite their well known significance, the current methods for software architecture design (e.g., [9], [10]), as well as pattern languages (e.g. [11]) don't support the architect in making a choice among different technology solutions. Alternatively, they focus on selecting a combination of conceptual solutions as first class concepts for solving the different design issues, presupposing a direct mapping from the conceptual solutions to the technology solutions [12].

Our main goal in this paper is to support the architect taking technology design decisions, through answering the following research questions:

1) *RQ1: How does the architect conceive software technologies as architectural solutions during the decision making process?*
2) *RQ2: How can we model and relate technology decisions with existing architectural knowledge concepts?*

In order to answer these questions, we conducted an explorative research study. We started with a qualitative content analysis research process, followed by refinement and validation interviews. We analyzed the different perspectives, which

978-1-4799-1922-2/15 $31.00 © 2015 IEEE
DOI 10.1109/WICSA.2015.14

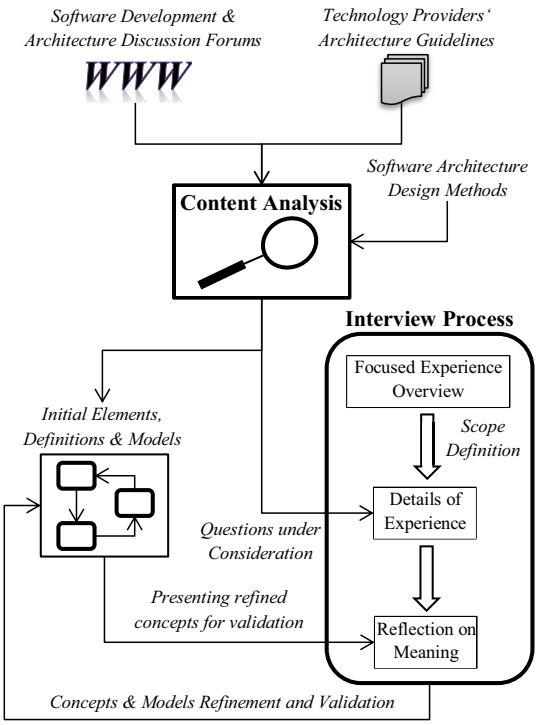

Fig. 1: Research Process Diagram.

the technology vendors and architects have in offering and choosing technology solutions respectively. Our contribution is modeling technology solutions as a set of features, which are offered by the technology vendors, as well as their associated architecturally significant technology aspects (ASTAs), which are considered by the architect in taking the technology design decision. Finally, we integrated both concepts with existing architecture knowledge and ADDs models.

The rest of the paper is structured as follows. In Section II, we describe and explain in details our research process. In sections III, and IV, we present our defined concept of modeling technology solutions as a set of features and aspects. The two sections are followed by Sec. V, where the proposed architecture knowledge model for technology decisions is proposed. In Sec. VI, our validation results for the interviews data anaylsis is presented, as well as a discussion about the threats of validity for our study. Related work in the current state of the art is presented in Sec. VII. Finally, Sec. VIII provides our conclusions and future work.

II. Research Process

In order to answer our research questions, we followed the research process shown in Fig. 1. We divided our research process into two main phases:

1) *Data Gathering and Hypothesis Definition*: In this phase, our main goal was to collect information about the technology decisions. We wanted to understand, what are the primary factors, which make an architect choose or reject a certain technology solution, and what are the scenarios the architect faces during a technology decision. In order to achieve this goal, we followed a qualitative content analysis research method among different technology resources. As a result of this phase, we formulated our initial hypothesis for the technology decision concepts and models.
2) *Hypothesis Refinement and Validation*: In this phase, we refined and validated our proposed concepts and models. It was important to align our understanding with what practitioners do in their work. Since the interview research method is the best method in discovering the human experience [13], we conducted a set of interviews with architects and experts, who are used to take technology decisions frequently. This process helped us to improve our model, as well as to validate our ideas.

In the following sections, we explain both the content analysis and the interview research processes respectively.

A. Content Analysis

Technology selection guidelines and reasoning are explained in different sources:

1) *Technology Vendors Architecture Guidelines*: Each technology vendor provides guidelines (e.g. [14], [15]) for designing software systems using their designed products. However, they do not conduct comparisons between products from different vendors, to show their strengths and drawbacks.
2) *Technology Discussion Forums*: This is a rich source for exploring, how technologists choose a technology solution. The discussions show the factors, which drive an architect to choose a certain technology. In addition, comparisons between technologies are usually part of the discussions. For example, on the stackoverflow forum, you might find a topic with title "Spring MVC vs JSF". Nevertheless, technology forums lack information about the design reasoning and processes, and their relationship to the architect concerns in taking a technology decision.
3) *Software Architecture Design Methods*: Even though most of the architecture design methods don't provide detailed guidance on taking a technology decision, some architecture design methods (e.g. [6]) provide guidelines on the different situations, that the architect can face during the selection of technology solutions.

We followed a qualitative content analysis [16] text analysis process through several selected resources from each of the above mentioned categories. We selected our data analysis sources based on their popularity, and richness in technology decision knowledge. We followed several guidelines in text analysis, such as coding, and memoing to refine and categorize the text. The combination of different resources analysis supported us to propose our hypothesis from different perspectives.

TABLE I: Interview Participants Experience Overview

ID	Exp. (Years)	Technology Background	Role	Industries
1	10	C/C++, Microsoft Products	Technology Consultant	NLP, Performance Critical Systems, E-Commerce
2	13	Microsoft Technologies	Architect	Flight, Communications, Social Media, Reservation, Retail and Education.
3	11	Java / J2EE	Technology Consultant	Billing, Medical-care, E-commerce
4	9	Java / J2EE	Enterprise Architect	Telecom, Costing & Billing, Oil & Mining, Military
5	10	Java / Integration Technologies	Technology Consultant	Communications, Transportation
6	7	Java / J2EE	Technology Consultant	E-Government, Automotive
7	12	Database Systems, Microsoft Products	Enterprise Architect	E-Government, Financial

B. Interviews

As we were seeking experience in technology architectural decisions, several factors have been considered in choosing the interview participants: 1) Their experience in using and choosing technologies. 2) Their architecture knowledge and design skills. 3) The size of the companies and systems, they are working in or have worked in before. 4) Their interest and motivation to participate in the study. Before choosing the interviewed experts, we identified more than 20 candidate experts through our personal connections. We evaluated them based on the mentioned criteria, to settle on the 7 participating experts. It is interesting to mention that, even though some of the candidate experts work in the role of a software architect in their companies, they do not take technology decisions. In these companies, the architecture decisions are divided between two different roles, the software architects, who design the system conceptually, and the technology consultants who choose the technologies. Therefore, we included in our study experts, who only take technology decisions.

The interviewed experts work, or worked before in software houses or IT service companies, with more than 100,000 employees. All the experts have either a Bachelor or Master degree in computer science or engineering. Due to the fact that the participants live and work in different cities, the interviews have been conducted remotely through telecommunication software.

We followed a three-phase interview process as proposed by Seidman [13]:

1) *Focused Experience Overview*: In this phase, we asked the interviewees to answer several questions to show their experience in software architecture, as well as their technology experience, projects and domain of work. We made this step one or two days before the first meeting, which supported us preparing the suitable questions which align with the participant's context and experience. Table I shows a brief summary for the experience overview of the participants.
2) *Details of Experience*: In this phase, we intend to learn from the participant's experience, in order to refine and validate our concepts. Our initial hypothesis, that we concluded from the content analysis phase helped us to direct our questions and discussions. In order to do this, we mapped each concept in the hypothesis with one or more interview questions, which has been customized based on the result from the first phase. During the interview, we were giving the space for the interviewees to explain and express their opinion, and tell us about their experiences, which was the main feedback in this meeting to verify and improve our concepts. The result from this meeting is a set of real practical examples from each participant's experience, which either align, or improve or contradict with our initial hypothesis. In the following sections, we are going to state some of the *Interview Questions (IQ)* that we asked to the experts, as well as their responses[1] and examples.
3) *Reflection on the Meaning*: After the first meeting, we were able to refine and verify our initial concepts, through the examples and discussions presented by the participants. However, it was important to validate our interpretation between the experience examples and the proposed concepts. Therefore, in the second meeting, we focused on discussing the proposed hypothesis concepts, and relating it to the mentioned experiences. First, we explained our research goal, and the initial concepts and models, and for each concept, supported by an example from the participant's experience, we asked the interviewee, if this concept align with their understanding and practice. Based on their feedback, we either validated or changed or rejected a certain concept. Sec. VI presents our evaluation results for the proposed concepts, based on the interviewees feedback.

For both interviews and for each participant, we recorded and transcripted the interviews, to allow us analyzing the discussion. The length for each interview was between 60 and 120 minutes. The difference in duration between the first and the second interview for each participant is between 2 and 7 days.

III. Technology Features

One of the main challenges during software architecture design is choosing the right technology solutions (e.g. COTS or frameworks), in order to implement the designed architecture. Even though different technology solutions act as alternative solutions for the same problem, they are different in their capabilities and qualities.

Technology vendors offer their proposed solutions as a set of *Technology Features*. These features are the abstract capabilities, which they claim, that these technologies provide. The vendors usually describe the merits of each feature, and how they could be used to implement a software system. Features could be classified with respect to the capabilities provided into different types. In the following paragraph,

[1]Some of the participants' answers have been translated from their native language to English.

we list and define the types of capabilities offered by the features. This list has been derived from our analysis and the interviewees' practical experiences:

1) *Development and Configuration Capability*: It provides the ability to develop or configure an implementation for a solution through a development environment, which comprises programming languages and possibly development and testing tools. An example from our analysis is the Microsoft Windows Communication Foundation (WCF) technology feature to develop services for a Service Oriented Architecture (SOA). The services development is done through the C# or VB programming languages, supported by the Microsoft development and testing tools, while, the services' protocols are dynamically defined through a set of XML configurations.
2) *Behavior Capability*: It provides either existing and compiled software components, which implement solutions with a certain quality, or a forecasted behavior for a possible development at certain conditions. For example, several web-based technologies (e.g. JSF, ASP.Net) provide an implemented web process, which embodies HTTP requests handling, and HTML generation functionalities. On the other hand, programming languages' compilers provide different behavioral capabilities, when compiling the source code to object code. For example, the ability of the Java virtual machine compilers to handle multithreading source code among different platforms.
3) *Usability Capability*: It provides either an existing user interface functionality, or facilities for developing a user interface. The main target from these capabilities is to facilitate the usability of the developed system. For example, Microsoft Sharepoint provides out of the box user interface features for content and document management. On the other hand, the recent version from HTML 5.0 supports the web designer to use more elegant forms, as well as additional interactive user interface features (e.g. drag and drop) to support an easier front-end development.
4) *Interoperability Capability*: It provides the ability for the technology solution to integrate and communicate with other technologies. The features could be either through an implemented interface, or through supporting a well-known standard or protocol (e.g. SOAP). For example, Java technologies could access Microsoft SQL Server through the JDBC SQL Server Data Access component. On the other hand, Microsoft WCF offers implementations for 9 protocols (e.g. HTTP, TCP/IP, P2P, ...).
5) *Storage Capability*: It provides the ability for the technology to store data, considering the data size, format and processing. For example, Oracle database offers different product editions, the standard edition supports storage with maximum 11 GB, with a single CPU processing, while the enterprise edition has no storage or processing limit.
6) *Operational Capability*: It provides the ability for the technology to monitor and manage the processing of the system during execution. For example, HP Openview products offer features for application status, HW resources, and database system's monitoring.
7) *Commercial Capability*: It is concerned with the price, licenses, and vendor or community support for this technology solution.

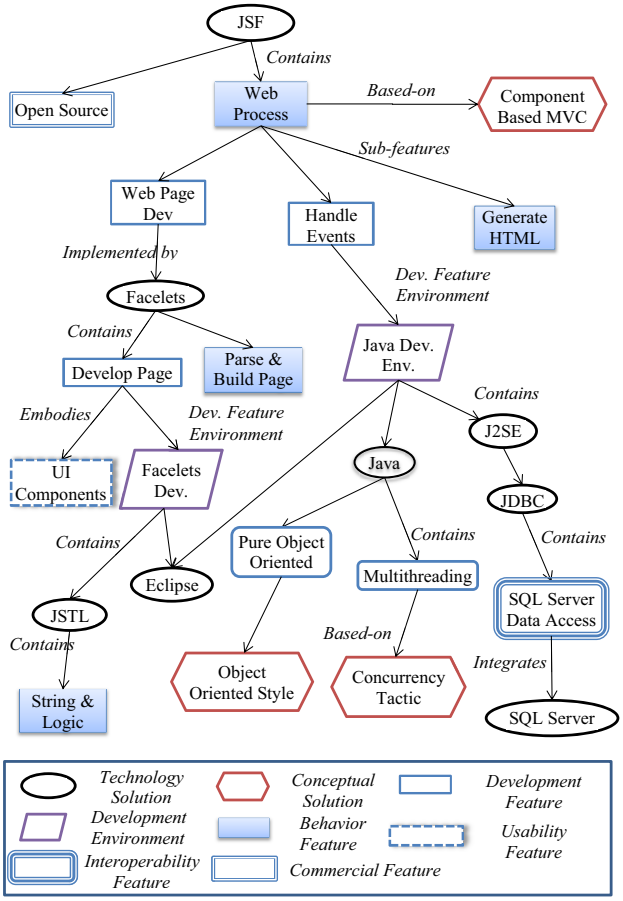

Fig. 2: An example for a technology features' tree.

Each technology solution encapsulates a tree of interrelated features. Each feature provides a capability from the previously defined capabilities' types[2]. Fig. 2 shows an example for a partial technology features' tree, which has been created through our content analysis activity, integrated with the industrial examples mentioned by the interviewed experts. For example, Java Server Faces (JSF) is an open-source technology solution *'Commercial Feature'*, to support developing a web based application. It contains a web process *'Behavior Feature'*, which is based on the component-based MVC pattern. Several sub-features are associated with the parent web process feature. This includes developing the user interface, handling events *'Development Features'*, and generating HTML *'Behavior Feature'*. Even though, JSF offers development user interface feature, it depends on another technology solution "Facelets" to implement this feature, which consequently offer a development feature, through a development environment. In addition, the Facelets develop-

[2]We will refer to features, which provide a certain capability type with the capability name, e.g. Development Feature, Behavior Feature

TABLE II: Capabilities' Types and Architectural Concerns Relationships

Capability Type	Influenced Architectural Concerns
Development and Configuration Capability	Development Time, Training Time, Maintainability , Testability, Configurability, Evolvability
Behavior Capability	Performance, Reliability, Security, Accuracy, Portability, Reusability
Operational Capability	Manageability, Supportability, Availability
Interoperability Capability	Interoperability, Inter-process communication
Usability Capability	Usability
Commercial Capability	Cost of Ownership, Openness, Development Time, Training Time
Storage Capability	Data Accessability, Scalability

ment feature provides a set of user interface components *'Usability Feature'*. On the other hand, JSF event handling development feature is provided through a Java development environment, which incorporates development tools, and the Java programming language with its pure object oriented style. The Java development environment contains several libraries, which embody many additional features. For example, the JDBC library offers an *'Interoperability Feature'* to access Microsoft SQL Server database technology.

Each of the technology capabilities influence one or more of the architectural concerns, and subsequently each of the offered technology features influences different concerns based on its adopted capability type. Table II shows the relationship between each of the identified technology capabilities' types and the different architectural concerns [17], [18]. Understanding the influence of the technology solutions' features on the different concerns supports the architect to evaluate and compare the different technology features, and consequently justify the technology design decision. In Sec. IV, we explain how the influence of features on the stakeholders' concerns differentiate the technologies from each other.

IV. Architecturally Significant Technology Aspects

As explained in the previous section, technology solutions embody many different features, which are designed and implemented within the technology. Consequently, it was interesting for us to ask the participants, "*IQ: To what level does the architect need to know about technology features in order to take the right ADDs?*" Bass et al. [6] listed several important considerations in choosing a technology solution, such as the capabilities of the development tools, the familiarity of the development community with this technology, the possible vendor and community support, the drawbacks of the technology, and the compatibility of this technology with the existing technology stack.

One of the interview participants mentioned that *"The architect doesn't need to know the technology in depth. However, he needs to know the differences between the different technologies, their benefits and drawbacks, regarding performance, vendor support, ..."*, another answered *"He needs to know how technologies work from a high level, considering its learning curve, development effort, usability, ..."*.

Based on our analysis and the interviews' discussions, we define in this section the different types of *Architecturally Significant Technology Aspects (ASTAs)*. They are the principal and distinctive technology solution's characteristics, which distinguish the technology solution from other alternative solutions, and consequently support or influence the architectural design decision. In other words, these aspects qualify such a technology solution to be selected by the architect to satisfy one or more ASRs.

ASTAs could be considered as either *Benefits* or *Drawbacks*. Benefits are the advantages, which the technology solution have over other competitive solutions. On the other hand, drawbacks are either technology features, which are missing in this technology solution, even though they exist in other alternative solutions, or they are well-known existing features problems, which need to be considered by the architect during the decision making. Both benefits and drawbacks act as different sides of the same coin, such that the same technology solution benefit could be a drawback in another solution. Moreover, both types act as important factors for an architectural design decision.

Each ASTA is associated with a technology feature, and consequently have the same capability type as their related features[3]. One of the interview participants mentioned *"Even though Java provides an important feature for code portability among different platforms, this is a significant drawback for our development, which seeks native components development. This makes us always favor C as our development technology. Nevertheless, it lacks such a platform independent feature"*. Furthermore, we believe that both benefits and drawbacks are relative notions, which could be solely determined through a comparison with other competetive technologies. Such a comparison are not usually provided by the technologies' vendors. However, they are part of the software community discussions and experiences.

Due to the fact that features are commonly described ideally by the technology vendors, without mentioning their drawbacks, several discussions on the technology forums try to share their experiences with either problems, which they faced in using these features, or missing capabilities, which were expected to be provided. Associated with these discussions are side-by-side comparisons between features from different technologies, which show the benefits and drawbacks of each feature in comparison to the other technology features. We call these types of aspects *Feature-Based ASTAs*.

One of the interview participants mentioned the following: *"In order to choose a web-based framework, we depended on a comparison between Spring MVC and other frameworks. We prefered to use the Spring MVC framework over other frameworks, because it's supported with better documentation, which make it easier to develop and learn."* In this situation, the interviewee took the design decison based on a development feature-based benefit, which is the "better development documentation".

On the other hand, as we mentioned in Sec. III, each feature influences different types of architectural concerns. This influence could be measured differently depending on the type of feature's capability and the assessed concern. For

[3]We will refer to ASTAs, which assess a certain capability type with the capability name, e.g. Development ASTA, Behavior ASTA, ...

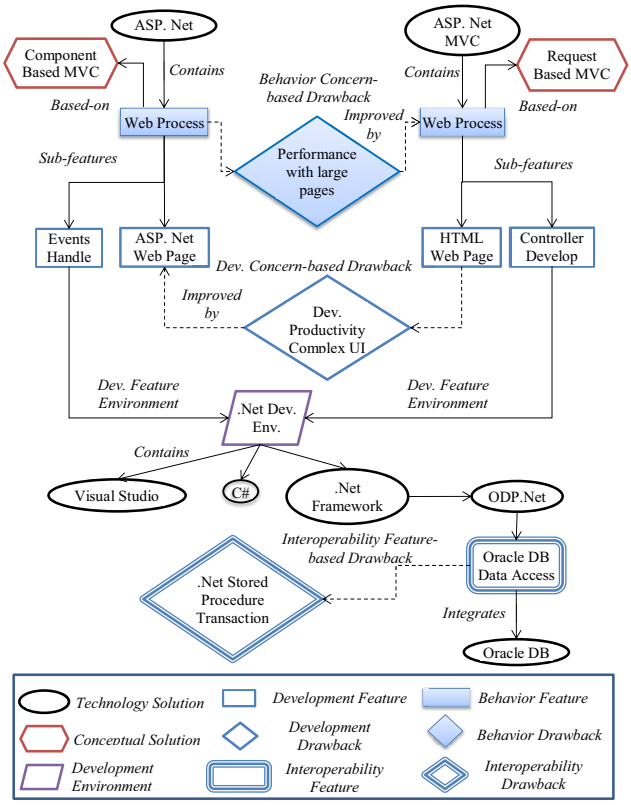

Fig. 3: A subset from a technology features' tree with both feature and concern based drawbacks (ASTA).

example, benchmarking can be a suitable method for comparing the performance of the behavioral features for multiple technology solutions. In addition, development features could be measured through development productivity experiments (e.g. [19]). As a result of these measurements, we could recognize, which technology solution features are better than the others regarding the measured concerns. In other words, the concern-based benefits and drawbacks or *Concern-Based ASTAs*.

An interview participant described an experience about a certain database management system *"After 1 year, the amount of data in the database reached more than 20 billion records. After several testing, we discovered that the performance of data processing operations is degrading exponentially with the increase in the amount of data, this technology drawback derived us to take an architectural decision to replace such a database management system"*.

Fig. 3 shows two web interface technology solutions, ASP.Net and ASP.Net MVC. Both technologies are based on the MVC architectural pattern. However, they are based on different pattern variations. This difference triggers a distinction in their development and behavior features. An interviewee mentioned *"According to our experience, ASP.Net provides a feature for faster and easier development environment than the ASP.Net MVC, due to the availability of many reusable UI components. However, ASP.Net is slower than ASP .Net MVC in handling thousands of users, due to the fact, that Component-based MVC need to store the view state of each UI control."* By analysing this statement, we could identify the following ASTAs: A) ASP.Net faster and easier development than ASP.Net MVC is a *'Development Concern-Based Benefit'*, B) The availability of many reusable UI components in ASP.Net is a *'Development Feature-Based Benefit'*, C) ASP.Net is slower than ASP.Net MVC in handling thousands of users is a *'Behavior Concern-Based Drawback'*, and D) Component-based MVC need to store the view state is a *'Behavior Feature-Based Drawback'*.

From this experience example, we can conclude that both types of ASTAs are strongly related, such that an identified feature-based drawback would impact the feature negatively, which might lead to a concern-based drawback. Similarly, a feature-based benefit would impact the technology feature positively, which possibly lead to a concern-based benefit.

V. Architecture Knowledge for Technology Design Decision

In this section, we consolidate and model the technology features and ASTAs' concepts, which are presented in the previous sections. In addition, we integrate the proposed technology concepts with the reusable design decision concepts proposed by Zimmermann et al. [20], [21]. Finally, we appended additional decison making elements to formulate our proposed architecture knowledge (AK) model. The proposed model supports the architect to make the technology design decision.

Fig. 4 shows the meta-model for the proposed architecture knowledge. At the core of the model are the design issues, which are the architecture design problems facing the architect. Each design issue is associated with a set of alternative architectural solutions. An architectural solution could be either a conceptual solution, or a technology solution, or a feature within a technology solution. As explained in the previous sections, each technology feature may have certain ASTAs, which are identified through comparisons between different technology features. ASTAs could be either benefits or drawbacks, feature-based or concern-based. Moreover, the identified ASTAs are part of the technology decision justification.

A feature's drawbacks are usually sources of additional design issues, which the architect need to overcome before selecting the technology solution. Overcoming solutions are different, however, one of their options is using other technologies, which contain features that address this problem. For example, one of the interview participants mentioned *"JSF framework has drawbacks, which impact the usability quality attribute. Two options were proposed, either to use an additional framework (e.g. Primefaces) to mend this problem, or hire a web designer to fix the problem manually"*.

During our interviews, we asked all the participants the following question: *IQ: What are the steps you take to choose between a list of possible technology solutions?* It was not surprising that each participant described a different reasoning process than the others. However, all participants shared a common set of elements, which they consider in selecting the technology solution, even though, they are used in a different order. Therefore, the proposed AK model contains

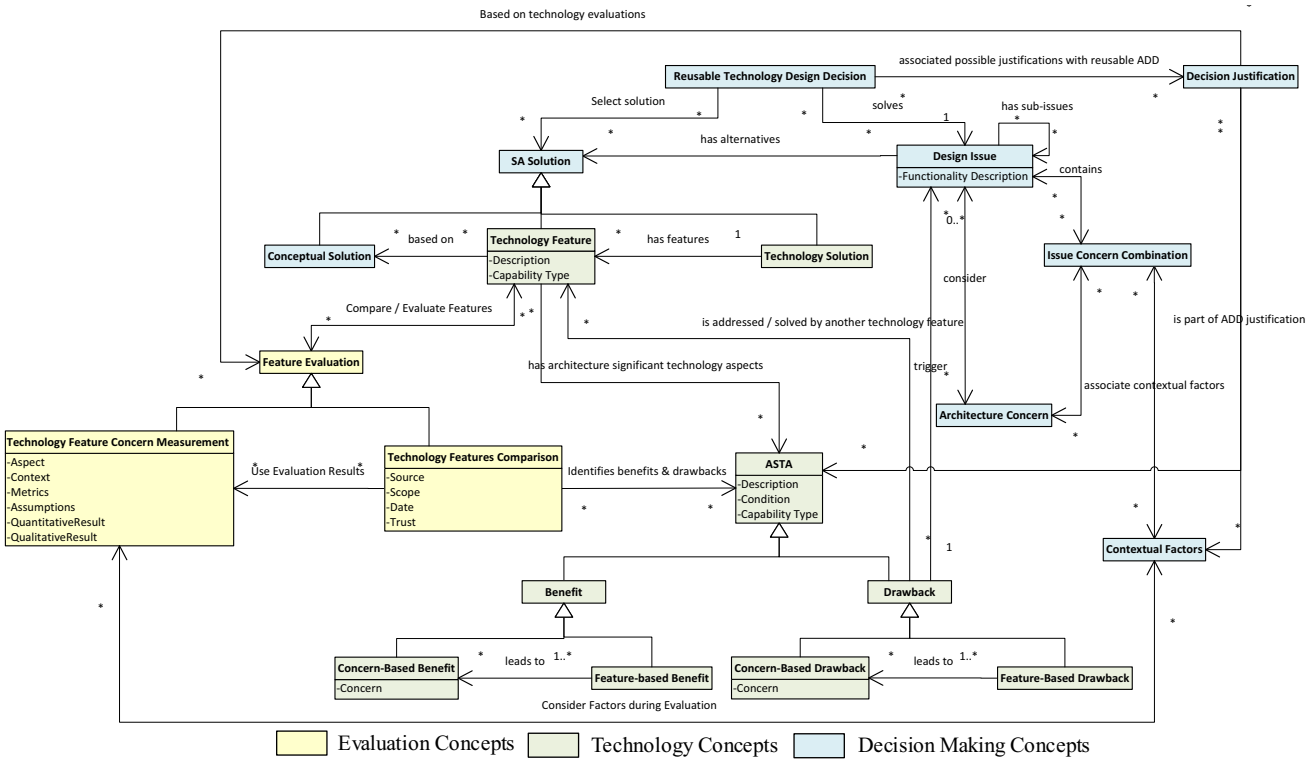

Fig. 4: Software Architecture Knowledge (AK) Metamodel.

and relates the different elements, which are necessary to assist the architect in taking a technology design decision, such that it can align with the different design reasoning methods.

Even though, each of the interview participants described different steps in choosing a technology solution, we can still group their description into two main reasoning types [22], [23]:

1) *Deductive, problem-driven*: In this process, the architect starts by analyzing the design issue, and its associated architectural concerns. For example, in choosing between different middleware technologies, interoperability, performance, security and development time should be considered. For each of these concerns, several factors are associated; the technology at the client and server, the size and structure of the transfered data, the network between the two sides, and the development team skills respectively. Based on these factors, the architect can start evaluating the different technology features (e.g. [24]). Alternatively, the architect could assess the feature's quality through evaluating the conceptual solution (e.g. evaluation for architectural patterns [25]), which this feature implements. By the end of the process, the architect needs to make trade-offs among the different concerns (e.g. using [26]).
2) *Inductive, solution-driven*: In this process, the architect starts by checking other experiences, which have similar situations. In other words, design decisions which have been taken in different projects but with similar circumstances, and based on matching both conditions, the architect chooses the suitable technology solution. We believe that technology decisions that are justified based-on the architectural concerns and factors are reusable, such that similar architectural concerns and factors could be repeated among different projects. On the other hand, technology decisions that are justified based on business or social aspects cannot be reused within the context of software design decision (maybe arguably reused in a business context).

Even though our main research goal is not to drive a design process, it is important to understand the different reasoning methods, which the architect uses, in order to identify the important architecture knowledge elements and their relationships.

VI. Evaluation Results

A. Interview Responses Analysis Results and Observations

Table III shows the data analysis results for the interviewees' responses. In order to accurately evaluate the feedback of the participants for each of the explained concepts in the previous sections. We designed several levels of responses:

1) *Concept Contribution*: The participant mentioned based on his experience a new concept or an improvement to a concept which was not originally part of the content analysis derived hypothesis.

TABLE III: Interview Data Analysis Results
++: Concept Contribution, ✔: Concept Supported
Y: Concept Accepted, N: Concept in Doubt
–: No Answer Provided

Concept vs. Participant	1	2	3	4	5	6	7	%
Technologies as Features & ASTAs (Sec. III & IV)	✔	✔	✔	✔	✔	✔	✔	100
Development & Configuration Feature or ASTA (Sec. III & IV)	Y	N	✔	++	✔	✔	Y	86
Behavior Feature or ASTA (Sec. III & IV)	✔	Y	✔	✔	✔	✔	✔	100
Usability Feature or ASTA (Sec. III & IV)	Y	Y	Y	✔	Y	✔	✔	100
Interoperability Feature or ASTA (Sec. III & IV)	–	++	Y	Y	++	–	✔	71
Storage Feature or ASTA (Sec. III & IV)	✔	Y	Y	Y	Y	–	✔	86
Operational Feature or ASTA (Sec. III & IV)	–	–	–	–	++	Y	✔	43
Commercial Feature or ASTA (Sec. III & IV)	Y	Y	++	Y	Y	–	✔	86
Decision Making Factors (Sec. V)	Y	✔	✔	Y	✔	–	Y	86
Decision Making Concerns (Sec. V)	✔	Y	✔	✔	✔	–	✔	86
Decision Making Evaluation Report (Sec. V)	Y	–	✔	–	–	✔	✔	57
Decision Making Deductive Problem Driven (Sec. V)	Y	✔	✔	✔	Y	Y	✔	100
Decision Making Inductive Solution Driven (Sec. V)	Y	–	–	–	✔	✔	✔	57
Drawbacks ASTA (Sec. IV & V)	✔	Y	++	Y	++	✔	✔	100

2) *Concept Supported*: The participant supported the hypothesis concept with additional examples from his experience.
3) *Concept Accepted*: The participant accepted the proposed concept. However, she does not have an example from her experience to support it.
4) *Concept in Doubt*: The participant indicated that the concept is unclear to her, or it does not align with her experience.

The concepts which were characterized as unclear by the majority of the participants have been removed.

All the concepts have been experienced by the participants during their decision making experience. We can observe that the drawback ASTAs concept (Sec. IV & V) and the interoperability features and ASTAs (Sec. III & V) were the mostly contributed concepts due to their importance to the participants. However, some of the contributed concepts have not been evaluated by all participants, due to the fact, that the interviews have been conducted incrementally.

B. Threats to Validity Assessment

As all qualitative empirical studies, our study faces validity threats. This section explains the construct, internal, and external validity threats, as well as reliability of the study.

1) Construct Validity: In this type of validity, we are concerned with validating the accurate representation of the initial content analysis hypothesis through the interviews' questions, as well as the interviews' answers interpretation. In order to map our initial hypothesis to the interview questions: First, we defined a set of general questions, such that each question is related to a hypothesis concept using a concept mapping. After our first phase of the interview process, we were able to adapt these questions to align with the experts understanding, which supported a more suitable explication of the hypothesis constructs. The interviewee had no idea about our initial hypothesis during the second phase of the interview process. In addition, the interview participants work in different companies, and they don't know about each other. This prevented any interactions between the different experts, as well as any possibility of hypothesis guessing.

During the third phase of the interview process, our main goal was to validate our interpretation for the experiences and examples collected during the second phase of the process. By presenting and explaining our concepts and relating it to the examples mentioned by the participants, we were able to assure that we have the right generalizability across the hypothesis constructs. In addition, conducting the concept validation among all the participants supported us to minimize biasing during the results interpretation. However, we based our concept discovery and validation on 7 interviews, which is insufficient to cover all the possible aspects of technology decisions. Therefore, we believe that additional empirical studies are needed to extend the proposed concept.

2) Internal Validity: In this type of validity, it's important for us to insure that the interview setup supported us to drive the concluded results. In conducting our interview, we followed a set of guidelines (e.g. [27]) in questions preparation, as well as in managing the conversation with the participant. With each participant, we started with a general question, like *"IQ: What are the factors which influence choosing a technology?"* during the participant answer, we give the freedom for the expert to explain his answer, and we asked the expert to focus on real experience examples. This supported us to interpret the meaning. In some cases, we mentioned the same question twice, however, with different ways to ensure that the participant provides the needed information, without interrupting his speaking. In addition, the 3 phase process helped us to have more than a chance to clarify our understanding to the examples or concepts explained by the interviewees. Even though, it was originally assumed that all experts should have the same level of experience, five of the interviewed experts contributed to the concepts more than the other two. However, this ratio shouldn't impact our results. Moreover, all experts supported us in the model validation through the interview "reflection of meaning" phase.

3) External Validity: In this type of validity, we would like to assess our interview study results regarding its generalizability. Regarding this aspect, we have several threats of validity. We did not select the interview participants randomly, as we depend on our network of experts. However, we made no control on their mentioned experiences. As the different participants have experience in different domains, we didn't focus our discussion on domain specific problems. Even though one of the participants has experience in embedded systems, the interview focused on experiences and technologies used within information and distributed systems domain. Moreover, we focused on the solution level of software architecture during our discussion, more than the enterprise level.

4) Reliability: In order to support the reliability of measurement, we followed an intra-rater reliability method, such that the 3rd phase of the interview ensured that the same

concepts have been validated and evaluated by the interviewed participants, while the participants responses were recorded by a single interviewer. All the interviews were conducted one-to-one with prepared questions, which give the chance for each participant to give his opinion about others' inputs.

VII. Related Work

A. Software Architecture Design Methods

In the past two decades, several prominent software architecture design methods (e.g. RUP 4+1 views [28], [29]) have been suggested and utilized in practice. The proposed methods target modeling the software architecture in several views, such that each view comprises distinctive diagrams for modeling the proposed solution, in order to satisfy the stakeholders' architectural concerns. In addition, the methods provide several guidelines for the correlation between the different viewpoints. Most of the methods dedicate a viewpoint for the implementation or system realization. For example, one of the RUP 4+1 views is the the development view, which model the proposed solution technology components and connectors, providing guidelines for taking the technology decisions, such as ease of development, software management and reuse. However, the proposed architectural methods provide minimum support for a concrete architectural knowledge base [30]. Consequently, this makes the architects depend on their personal experience, instead of reusing and learning from others experiences.

B. Pattern Languages

A pattern language is concerned with defining a group of patterns, which solve related problems in the same domain. Moreover, some pattern languages provide relationships between patterns, to support the architect taking the design decisons through moving from one pattern to another. In the past two decades, many pattern languages have been proposed (e.g. [31]). However, each pattern language addresses a different domain of problems. Typically, pattern languages do not incorporate technology solutions as first class elements within their network of decisions. Nevertheless, each pattern provides optionally a list of technology examples, which implement this pattern. A proposed pattern language [3] integrate technology solutions with pattern languages. The proposed language model interrelationships between different technology solutions, as well as an implementation relationship between technologies and patterns. However, the suggested solution's network doesn't provide guidance for decison making and reasoning on technologies.

C. Software Architecture Design Decisions

Since the paradigm shift [2] of perceiving the software architecture as a set of architectural design decisions (ADDs), many approaches have been proposed. Even though all approaches are centered around the design decisions notion, they tackle different problems. A recent survey [32] on the architectural decisions field shows that most of the approaches focus on modeling, documenting and capturing the ADDs rationale (e.g. [33], [34] and [35]) for the sake of minimizing the software architecture erosion phenomena. However, these approaches don't support the architect in reasoning about the design decisons. Alternatively, other approaches (e.g. [36], [37]) consider the design decision as a mean for reasoning about the design problems. In other words, they try to model and depict the different methods, that the architect can use to think about an architecture design problem. Nevertheless, they don't support a concrete reusable architecture knowledge.

A recent study on the software architecture knowledge [38] shows that, the current architecture knowledge approaches have less support regarding architecture knowledge sharing and reuse. Differentiating the reusable architecture knowledge from the project specific have been considered by Ali Babar et al. [39] and Zimmermann et al. [20], [21]. The former approach considered a generic knowledge component based solely on patterns, which can be selected during the design decisions capturing. On the other hand, Zimmermann et al. proposed a reusable architecture knowledge framework, with the design issues and architectural solutions as the main elements, while the possible decisions' relations are modelled between them. In addition, the decisions are divided based on their granualarity into conceptual, technology and products. Even with the provided support for modeling technology decisions, the approach proposed by Zimmermann et al. lacks the ability to distinguish between the different technologies' capabilities from each other, such that it's hard for the architect to choose the suitable technology solution for the project situation. We believe that the approach proposed by Zimmermann et al. is promising regarding architecture knowledge reuse. Therefore, we based our approach on this proposed model.

VIII. Conclusion and Future Work

We started our study with the goal of understanding technology decisions in practice. Based on this goal, we designed our research method to provide maximum support for data gathering and validation from practitioners. Our contribution in this paper is modeling technology decisions, their features, architectural aspects, and evaluations as extension for existing architecture knowledge models based on empirical evidence from interviews and literature. The proposed approach considered the different design reasoning approaches that the architect use, as well as the different types of technologies and decisions.

In terms of future work, we are planning to implement the proposed AK models in a tool, and conduct an empirical experiment among practitioners, in order to assess and validate the usefullness of the approach. One of the concerns raised by the interviewees is the amount of knowledge required, and the complexity of their gathering and update. Thus, we believe approaches for technology knowledge exploration and extraction are needed, through natural language processing approaches among current technology literature and discussion forums.

References

[1] L. Chen, M. Ali Babar, and B. Nuseibeh, "Characterizing architecturally significant requirements," *Software, IEEE*, vol. 30, no. 2, pp. 38–45, March 2013.

[2] A. Jansen and J. Bosch, "Software architecture as a set of architectural design decisions," in *5th Working Conf. on Software Architecture*, 2005, pp. 109–120.

[3] S. Mahdavi-Hezavehi, U. van Heesch, and P. Avgeriou, "A pattern language for architecture patterns and software technologies introducing technology pattern languages," in *Proceedings of the 16th European Conference on Pattern Languages of Programs (EuroPLoP)*. Conference Proceedings, 2011.

[4] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal, *Pattern-Oriented Software Architecture: A System of Patterns*, 1st ed. John Wiley & Sons, Jul. 1996.

[5] E. Gamma, R. Helm, R. Johnson, and J. M. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Softwaresystemen*. Addison-Wesley Professional, 1994.

[6] L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice*, 3rd ed. Addison-Wesley Professional, 2012.

[7] B. Boehm, "A view of 20th and 21st century software engineering," in *Proceedings of the 28th international conference on Software engineering*, ser. ICSE '06. New York, NY, USA: ACM, 2006, pp. 12–29.

[8] C. Miesbauer and R. Weinreich, "Classification of design decisions: An expert survey in practice," in *Proceedings of the 7th European Conference on Software Architecture*, ser. ECSA'13. Berlin, Heidelberg: Springer-Verlag, 2013, pp. 130–145.

[9] L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice*, 2nd ed. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2003.

[10] J. Bosch, *Design and use of software architectures: Adopting and evolving a product-line approach*. New York, NY, USA: ACM Press/Addison-Wesley, 2000.

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

[12] H. Cervantes, P. Velasco-Elizondo, and R. Kazman, "A principled way to use frameworks in architecture design," *Software, IEEE*, vol. 30, no. 2, pp. 46–53, March 2013.

[13] I. Seidman, *Interviewing as Qualitative Research: A Guide for Researchers in Education and the Social Sciences*. Teachers College Press, 2006.

[14] M. Cade and H. Sheil, *Sun Certified Enterprise Architect for Java EE Study Guide*. Pearson Education, 2010.

[15] M. P. . P. Team, *Microsoft Application Architecture Guide, 2nd Edition*. Microsoft Press, 2009.

[16] U. Flick, E. von Kardoff, I. Steinke, and B. Jenner, *A Companion to Qualitative Research*. SAGE Publications, 2004.

[17] P. Lago, P. Avgeriou, and R. Hilliard, "Guest editors' introduction: Software architecture: Framing stakeholders' concerns," *Software, IEEE*, vol. 27, no. 6, pp. 20–24, Nov 2010.

[18] International Standardization Organisation, "Iso/iec/ieee 42010 - systems and software engineering - architecture description," 2011.

[19] G. Phipps, "Comparing observed bug and productivity rates for java and c++," *Software: Practice and Experience*, vol. 29, no. 4, pp. 345–358, 1999.

[20] O. Zimmermann, J. Koehler, F. Leymann, R. Polley, and N. Schuster, "Managing architectural decision models with dependency relations, integrity constraints, and production rules," *Journal of Systems and Software*, vol. 82, no. 8, pp. 1249–1267, 2009.

[21] M. Soliman and M. Riebisch, "Modeling the interactions between decisions within software architecture knowledge," in *Software Architecture*, ser. Lecture Notes in Computer Science, P. Avgeriou and U. Zdun, Eds. Springer International Publishing, 2014, vol. 8627, pp. 33–40.

[22] A. Tang and H. van Vliet, "Design strategy and software design effectiveness," *Software, IEEE*, vol. 29, no. 1, pp. 51–55, Jan 2012.

[23] A. Tang, "Software designers, are you biased?" in *Proceedings of the 6th International Workshop on SHAring and Reusing Architectural Knowledge*, ser. SHARK '11. New York, NY, USA: ACM, 2011, pp. 1–8.

[24] I. Gorton, A. Liu, and P. Brebner, "Rigorous evaluation of cots middleware technology," *Computer*, vol. 36, no. 3, pp. 50–55, Mar 2003.

[25] S. Bode and M. Riebisch, "Impact evaluation for quality-oriented architectural decisions regarding evolvability," in *Proceedings 4th European Conference on Software Architecture, ECSA 2010*, ser. LNCS, M. Babar and I. Gorton, Eds., vol. 6285. Springer Berlin / Heidelberg, 2010, pp. 182–197.

[26] T. Al-Naeem, I. Gorton, M. Babar, F. Rabhi, and B. Benatallah, "A quality-driven systematic approach for architecting distributed software applications," in *Software Engineering, 2005. ICSE 2005. Proceedings. 27th International Conference on*, May 2005, pp. 244–253.

[27] S. Hove and B. Anda, "Experiences from conducting semi-structured interviews in empirical software engineering research," in *Software Metrics, 2005. 11th IEEE International Symposium*, Sept 2005, pp. 10 pp.–23.

[28] P. Kruchten, "The 4+1 view model of architecture," *IEEE Softw.*, vol. 12, no. 6, pp. 42–50, Nov. 1995.

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

[30] D. Falessi, G. Cantone, and P. Kruchten, "Do architecture design methods meet architects' needs?" in *Software Architecture, 2007. WICSA '07. The Working IEEE/IFIP Conference on*, Jan 2007, pp. 5–5.

[31] F. Buschmann, K. Henney, and D. C. Schmidt, *Pattern-Oriented Software Architecture, Volume 4: A Pattern Language for Distributed Computing*. Chichester, UK: Wiley, 2007.

[32] 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.

[33] P. Kruchten, P. Lago, and H. Vliet, "Building up and reasoning about architectural knowledge," in *Quality of Software Architectures*, ser. Lecture Notes in Computer Science, C. Hofmeister, I. Crnkovic, and R. Reussner, Eds. Springer Berlin Heidelberg, 2006, vol. 4214, pp. 43–58.

[34] U. van Heesch, P. Avgeriou, and R. Hilliard, "A documentation framework for architecture decisions." *Journal of Systems and Software*, vol. 85, no. 4, pp. 795–820, 2012.

[35] D. Falessi, G. Cantone, and P. Kruchten, "Value-based design decision rationale documentation: Principles and empirical feasibility study," in *Software Architecture, 2008. WICSA 2008. Seventh Working IEEE/IFIP Conference on*, Feb 2008, pp. 189–198.

[36] V. Clerc, P. Lago, and H. van Vliet, "The architects mindset," in *Software Architectures, Components, and Applications*, ser. Lecture Notes in Computer Science, S. Overhage, C. Szyperski, R. Reussner, and J. Stafford, Eds. Springer Berlin Heidelberg, 2007, vol. 4880, pp. 231–249.

[37] A. Tang, M. H. Tran, J. Han, and H. Vliet, "Design reasoning improves software design quality," in *Proceedings of the 4th International Conference on Quality of Software-Architectures: Models and Architectures*, ser. QoSA '08. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 28–42.

[38] R. Weinreich and I. Groher, "A fresh look at codification approaches for sakm: A systematic literature review," in *Software Architecture*, ser. Lecture Notes in Computer Science, P. Avgeriou and U. Zdun, Eds. Springer International Publishing, 2014, vol. 8627, pp. 1–16.

[39] M. Babar, I. Gorton, and B. Kitchenham, "A framework for supporting architecture knowledge and rationale management," in *Rationale Management in Software Engineering*, A. Dutoit, R. McCall, I. Mistrk, and B. Paech, Eds. Springer Berlin Heidelberg, 2006, pp. 237–254.