The Software Architecture Mapping Framework for Managing Architectural Knowledge

Sébastien Adam, Alain Abran
Department of Software and IT Engineering
École de technologie supérieure
Montréal, Canada

***Abstract*—Within a software architecture design (SAD) project, designers deal with software design artifacts (SDAs) such as scenarios, patterns, and tactics. Each SDA has its unique issues and related architectural knowledge (AK) that may threaten the success of a project. This paper introduces the Software Architecture Mapping (SAM) framework to manage AK and associated issues by using finer-grained SDAs and networks of weighted arguments. These networks of data may be used to produce quantitative information in multi-dimensional views to facilitate the identification of critical SDAs and issues in a project. This paper illustrates how the SAM framework has been used to manage AK related to the template method (TM) design pattern in the context of an academic case study.**

Keywords: architectural knowledge (AK), architectural knowledge management (AKM), decision support systems, multi-dimension analysis, software design artifact (SDA), software architecture design (SAD), software architecture mapping (SAM), software structures map (SSM)

I. INTRODUCTION

During development of a software system, development teams deal with numerous software design artifacts (SDAs) such as scenarios, design patterns, and procedures. Each SDA can be characterized using a set of related SDAs and issues that are factors of influence that may threaten the success of a project. For instance, a design pattern [1] is an SDA characterized by a rationale, a solution, plausible consequences, and trade-offs that need to be considered for implementation. SDAs and related issues are assets of architectural knowledge (AK) that embody decisions and trade-offs applied during the project.

For development teams to be efficient, the SDAs and related AK must be managed and shared in an efficient fashion. Indeed, designers must evaluate how the most influential SDAs impact the capacity of the system to satisfy stakeholder needs. Insufficient details about these SDAs and their relationships may lead teams to inappropriate or suboptimal decisions. Several approaches propose a process or a technique aimed at managing SDAs [1,2, 3, 5, 7, 8, 10]. These approaches usually focus on a subset of the SDAs involved in the design process and on a specific development perspective that is part of the process. However, there is a lack of studies that support a multi-dimensional view of the AK database and a methodical treatment of the SDAs and related AK (i.e., the related SDAs, issues, and arguments that influence the activities and dimensions of a software development project).

Designer expertise and experience remains the key element for identifying the critical factors of a project and their appropriate solution. This is true at different stages of the development process (analysis, design, and implementation) and for different project aspects such as budget, quality, and schedule. AK should be managed in an integrated and systematic manner to enable the development of decision support systems that: 1) offer support to identify, describe, and analyze relevant SDAs, issues, and arguments; 2) relate SDAs to their factors of influence; and 3) keep track of the adopted arguments and resolved issues.

This paper proposes the Software Architecture Mapping (SAM) framework to manage AK and support multi-dimensional analysis of the AK database. The proposed AK model defines the following concepts: SDA, issue, software structures map (SSM), factor, argument, argumentation, and view. The SAM framework supports a two-phase approach for identifying, describing, and analyzing critical factors related to SDAs for a given project. The first phase is the assets creation phase, which aims at classifying the SDAs into one or more SSM and describing the related AK in the form of issues and interrelated arguments. The second phase is the assets consumption phase where the AK is used to provide views that facilitates identification of a project's critical factors.

The SAM framework has been applied in industrial contexts (software cockpit design and Web engineering) and academic contexts (catalogs of styles, patterns, and tactics, undergraduate design courses, and Web engineering) to evaluate its technical feasibility and usability, especially for novice designers. In addition, a requirements self-assessment has been conducted using the requirements for AK management proposed in the literature (e.g., architectural documentation rules [2]). As an example of SDA, this paper presents the template method (TM) design pattern [1] and related factors, which are analyzed and ranked to produce multi-dimensional views that highlight the critical factors of the case study. The contributions of this paper include: 1) reusable definitions of the AK model's constituents based on description formats for the SDAs, SSMs, issues, arguments, and argumentations; 2) a systematic method for executing a multi-dimensional analysis of the factors; 3) a flexible method to transform argumentations to multi-dimensional views.

DOI reference number: 10.18293/SEKE2016-183

The paper is organized as follows. Section II presents an overview of the SAM framework. Section III and Section IV illustrate the two phases of the proposed approach using a case study realized in the context of an undergraduate course. Section V presents related works and section VI the conclusions.

II. Overview of the Proposed Software Architecture Mapping (SAM) Framework

The SAM framework [11, 12] has been proposed to support software architecture design (SAD) and architecture knowledge management (AKM). The framework manages the AK defined by existing methods, models, and description templates from the literature on SAD and AKM [1, 2, 3, 7] (i.e., constraints, requirements, quality attributes, scenarios, concerns, rationale, styles, tactics, patterns, situational factors, assumptions, risks, components and connectors, fragments, viewpoints, views, procedures, metrics, and domain objects).

Figure I presents an overview of the SAM framework. The blank shapes represent the framework and the four basic concepts that constitute its reference model (i.e., the software design artifacts (SDAs), software structures map (SSM), argument, and view). The colored shapes are concepts of the Attribute-Driven Design (ADD) method [3].

The SAM framework defines two phases of knowledge processing: 1) asset creation is performed by a knowledge engineer, i.e., a software designer tasked with the creation of assets (i.e., SDAs, SSMs, arguments, and views); 2) asset consumption is performed by software designers that use the AK in the reusable assets of their projects. The asset creation phase elicits the factors that constitute the AK, followed by the asset consumption phase which analyzes these factors in order to create multi-dimensional views that enables identification of important factors of a software project. Each phase is independent. The results of the creation phase may be used for multiple executions of the consumption phase. Asset creation is organized into three steps: 1) identifying SDAs and related activities; 2) eliciting issues and impacted dimensions; and 3) describing arguments. The analysis phase is also divided into three steps: 1) selecting factors and building generic views for the SDAs under analysis; 2) ranking factors according to the context of the project and generating contextual views; and 3) identifying important factors of the project using the contextual views.

The two starting points in Figure I illustrate two ways to use the SAM framework. First, the SAM process may be used to acquire and share knowledge extracted from descriptions of styles, tactics, design patterns, and design decisions. Then, the resulting design knowledge base (i.e., SDAs and SSMs) may be used to support the SAD. At particular decision points in the design process, such as selection of a pattern, the software designers may use the SSMs of styles, tactics, patterns, or decisions as checklists of SDAs to elicit issues, describe arguments, and create views. For a specific decision point, an SSM may record the general, contextual, and design knowledge, and the arguments may record the reasoning, as proposed in the literature.

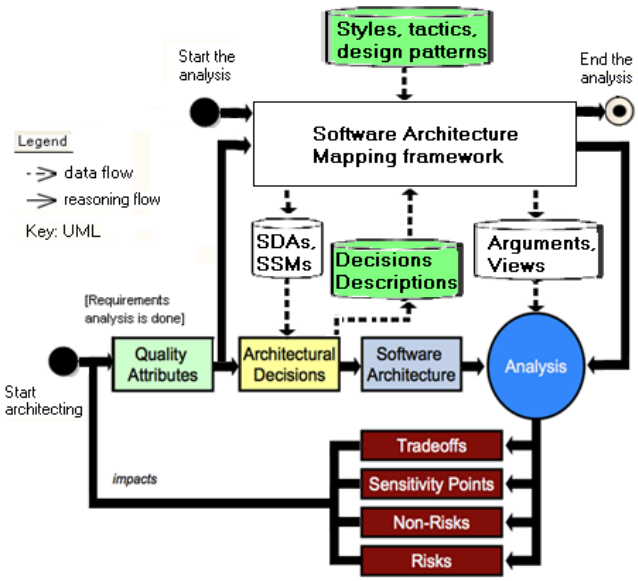


Figure I Overview of the SAM framework

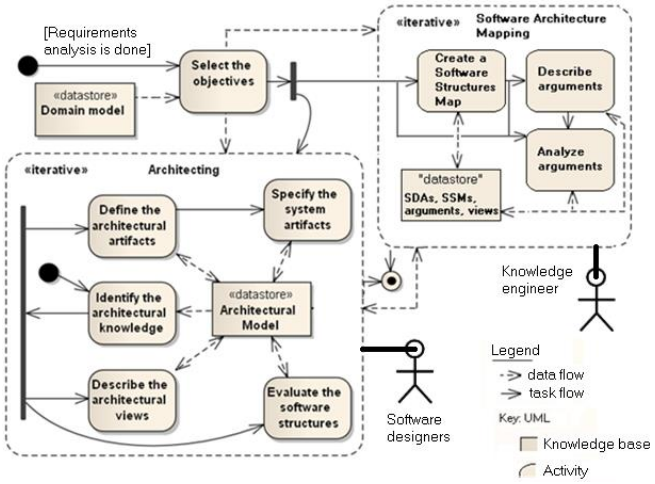


Figure II Overview of the SAM process

A. Process and roles of the proposed SAM framework

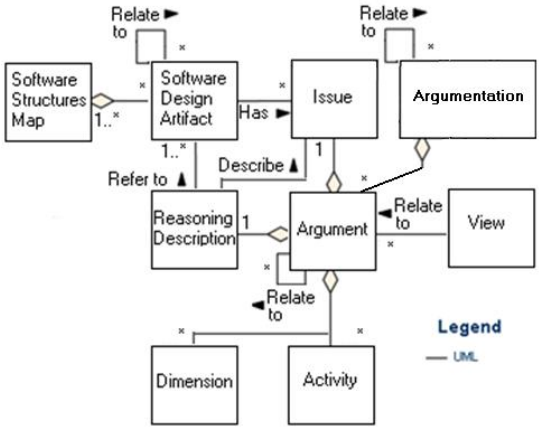
The proposed SAM process aims at managing the SDAs, SSMs, issues, and arguments related to a design. Figure II presents the three activities of the SAM process (i.e., create an SSM, describe arguments, and analyze arguments) and the task flow and data flow between the SAM process and the SAD. The activity "create an SSM" aims at 1) identifying the finer-grained SDAs related to either a decision point or the description of a style, pattern, or tactic, and 2) classifying these SDAs into a matrix of traceability. The activity "describe arguments" aims at 1) eliciting the issues related to the SDAs used and 2) reasoning about the arguments related to the issues. The activity "analyze arguments" aims at inferring the order of treatment of the arguments and issues based on the rankings and views of the AK repository created during the analysis.

B. The proposed architectural knowledge model

The proposed AK model is based on our previous work [11, 12], case studies, and controlled experiments applying the SAM framework. The AK model was developed by addressing the requirements and conclusions in the literature on methods, models, and tools for SAD and AKM. The model was designed to meet the following requirements: 1) capture rationale, constraints, design decisions, and related explications and quantifications about how they impact objectives; 2) reduce the possibility of expressing similar concepts with different terms; 3) take into account all activities and SDAs from the literature on SDA and AKM; 4) support personalization for context-specific SAD and AKM; 5) capture the SDAs and issues related to specific decision points; 6) capture the relationships between SDAs; 7) provide multiple perspectives for managing the AK repository; 8) support an integrated approach to SAD and AKM; 9) capture the AK from textual catalogs; 10) support selection and comparison of SDAs; and 11) support the evaluation of the SDAs and the consequences of applying each of them.

Figure III presents the concepts of the AK model for the SAM framework. AKM aims at sharing AK explicitly in a manner that supports AK evolution over time along with the architectures and their implementations. From our point of view, the SDAs and the arguments about their utilization constitute the explicit AK that relates to both the SAD and AKM. Each SDA has some related SDAs and issues. The SAM framework proposes to use: 1) the software structures map (SSM) for structuring the SDAs, 2) the argument for describing the issues and impacts related to the SDAs, 3) the view for analyzing the impact of arguments on dimensions and activities of SAD, and 4) the argumentation for structuring arguments. The SDAs describe the general context and design knowledge, and the arguments describe the reasoning. The following sections describe the concepts of the AK model.

Figure III The AK model of the SAM framework



The SAM framework defines a structure of software design artifacts (SDAs) for AKM. This structure of SDAs is the basic concept supporting the proposed approach. Many SDAs and relationships between them are described in the literature. An SDA may be but is not limited to: a tactic, a quality attribute scenario, a measure, a design pattern, a style, or any input or outcome of the SAD [12].

Definition: An SDA is any conceptual artifact that 1) provides design knowledge about the problem or solution spaces of a software design, and 2) corresponds to the identification heuristics of the SAM framework.

An SDA is either elementary or composite. The proposed definition is that an elementary SDA does not require the utilization of another SDA in the design solution, while a composite SDA does require the utilization of another SDA from the solution space where it is being used. For example, a tactic is an elementary SDA as proposed in [3], while a design pattern and a style are composite SDAs [1, 2]. The tactics described in [3] require no SDA from the solution space. The TM design pattern requires the utilization of the polymorphism tactic [1]. An SDA may have one or more applications, resulting in multiple descriptions of tactics [3], design patterns [1], and styles [2].

Definition: A software structures map (SSM) is a matrix of traceability that records a set of SDAs used either at a particular decision point during the SAD or in the descriptions of styles, tactics, and patterns.

The SSM is an instantiation of the classification scheme (CS) of the SAM framework [12]. The CS uses a matrix where the columns represent the interrogatives (why, when, what, which, how, and where) and the rows represent the activities of the SAD. The SDAs of the following activities occupy the row labels: select the objectives, identify the knowledge, and define, specify, describe, and evaluate designs. An SSM is managed as part of the AK. The SAM framework relies on the knowledge base of SDAs and SSMs for supporting the SAD. Related work [12] presents the table format used for representing an SSM. Each interrogative regroups only the SDAs classified into the corresponding column of the SSM. The SDA type gives the corresponding line of the SSM.

Definition: An issue describes a problem that occurs by introducing an SDA into a system being developed. One or more issues may be elicited for every change to a system.

The SAM framework defines a specific format to describe the issues. An issue description is composed of an SDA (subject), a verb, and a complement. The SAM framework proposes a list of verbs used for describing the issues [11]. The verbs capture abstractions that provide additional data about the issues, and express something that alters the meaning of the issue descriptions. Verbs support change from ad-hoc issue descriptions to predicate-issue structures. The verbs are used as a mean to facilitate the elicitation of issues and provide an issue description format.

Definition: A factor is an essential element for analysing how an SDA such as a design pattern, a tactic, or a style may impact a software design.

The SAM framework applies a multi-dimensional analysis using sets of factors that influence software engineering. A factor may be an SDA, issue, claim, reasoning, activity, or dimension. Table I presents the names and the descriptions of the six proposed factors.

TABLE I. FACTORS THAT CONSTITUTE THE ARGUMENTS

Name	Description
SDA	Software design artifact being examined
Issue	Problem that occurs by using or not using an SDA
Claim	Solution that occurs by using an SDA
Reasoning	Reasoning description about an issue or a solution
Activity	Set of cohesive development tasks
Dimension	Perspective on a set of evaluation results

TABLE II. THE PROPOSED ARGUMENTATION DESCRIPTION FORMAT

Argument: issue or claim, reasoning, and scope of the argumentation.
Reasons*:* arguments that support the claim.
Rebuttals*:* arguments that establish the falsity of the claim.
Alleviations*:* arguments that reduce the intensity of the claim.

Definition: An argument is a reasoned attempt to convince the audience to accept a point of view about an issue or a claim. An argument is an aggregation of factors, including at least one or more SDAs, one issue or one claim, and one reasoning description. The argument's scope may refer to impacted dimensions and activities.

A reasoning description describes the relationships between a set of factors; for example, between two SDAs – software designer and hook operation. The following reasoning description refers to two SDAs (software designer and object-oriented paradigm), an issue, and two dimensions (quality and people): "Using an object-oriented paradigm requires skills, expertise, and knowledge. The software designers do not master the object-oriented paradigm. This issue may impact the software quality and the software designers' commitment." The argument's scope refers to activities and dimensions strengthened (+) or weakened (-) by the argument. The activities are inferred from the activities related to the SDAs identified in the argument's reasoning. The dimensions are inferred from the dimensions impacted by the issue that prompted the argument. An argument is related to a dimension if there is a suspicion that the issue may produce variation (+ or -) of a dimension evaluation result.

Definition: An argumentation relates a set of arguments that describe how some SDAs create or resolve issues. Table II presents the argumentation description format. An argument provides the argumentation's claim, reasoning, and scope. Reasons, rebuttals, and alleviations are connection points. Reasons are arguments that support the claim. Rebuttals are counter-arguments for the claim. Alleviations are arguments that affect the claim.

Definition: A view is a matrix that puts into perspective a set of ranked arguments. A view analyzes an argument's impact on a design. In the SAM framework, the rows are labeled with activities such as designing, implementing, and managing, and the columns are labeled with dimensions such as functions, people, and quality.

The rankings of activities, dimensions, and arguments generate contextual views that are subjective and quantified. A views is used to identify critical factors of the project, which corresponds to the cell of a view that has a higher value. Cells are prioritized based on their values.

Cells with the highest priority (i.e., with a priority of 1) are used for reasoning further about factors that relate to the cell in order to nullify or reduce its value. Then, after these critical factors are addressed, the ranking are adjusted. The adjusted rankings provide new priorities. The analysis technique iterates these steps (i.e., identifying flaws and taking actions accordingly) until the user is satisfied with the values in the views (i.e., specific threshold values are attained).

III. ASSET CREATION PHASE

Section III illustrates the AK model's concepts and the phases of the SAM framework using the TM design pattern [1] (see [11] for more details).

A. Eliciting SDAs and related activities

Table III presents some of the SDAs identified from analysis of various TM descriptions given in the literature. Each SDA has a description and is classified under a specific type. We used the classification scheme and SDA types proposed in [12] as a means to facilitate elicitation of SDAs and constrain their interpretations.

TABLE III. DISTINGUISHING SDAS OF THE TM PATTERN

Software design artifact (SDA)			Act.
Id	**Type**	**Description**	
Ra1	Rationale	Define an algorithm, defer steps to subclasses	D
Pr1	Property	Object-oriented paradigm	AD
Pr2	Property	Reusability	AD
Pr3	Property	Extensibility	AD
Be1	Behavior	Template method calls primitive operations	DI
Op1	Operational	Define an abstract base class	DI
Op4	Operational	Define a template method	DI
Op5	Operational	Define a concrete child class	DI
Op8	Operational	Declare protected primitives operations	DI
St1	Structure	Abstract class	I
St2	Structure	Concrete class	I
Ro1	Role	Subclass writers	M
SF1	Situational	Multiple kinds of primitive operations	ADIM
Co1	Convention	Naming convention	IM

It is important to address each SDA during the activities that produce the most beneficial influences. An activity is a set of cohesive tasks intended to contribute to the achievement of a common goal. Table IV classifies each SDA of the TM pattern based on these criteria. We considered four important activities: architecting (A), designing (D), implementing (I), and managing (M) [11].

B. Eliciting issues

We analyzed the TM pattern to identify issues that may hinder its usage. Table IV lists some issues and the related SDAs. Due to lack of space, we present only a few of the numerous issues identified. Each SDA may solve or engender one or more issues. For example, the extensibility property (SDA) may not be well defined for a module (issue). Also, to lighten the responsibility of the subclass writers, the template method calls the primitive operations (SDA). Uncontrolled calls to primitive operations (issue) may cause problems. To address this issue, the pattern declares protected primitive operations (SDA). Our approach was to use a semi-formal argument format to describe the issues.

C. Describing arguments and impacted dimensions

One important objective of the asset creation phase is to describe arguments to use during the consumption phase to estimate the impact of each issue, which may differ depending on the context of use of an SDA. Argumentation is concerned with reasoning in the presence of imperfect knowledge by eliciting arguments for exploring issues rather than eradicating them [4]. In our approach, the argumentation was geared towards quantifying the impact of the factors on project dimensions and activities. The project dimensions we considered were adapted from [6] (also see [11]): Functions (**F**), Quality (**Q**), People (**P**), Budget (**B**), and Schedule (**S**).

Table VI presents some of the arguments we elicited to establish how each issue of the TM pattern impacts project dimensions (F, Q, P, B, S) and activities (M, A, D, I). For example, the argument (Arg1) predicts positive impacts on the functional dimension (F+) by declaring a final method. One reason is that a final method cannot be overridden. One rebuttal or reservation is that it is possible to hack the final mechanism (Arg7). The argument refers to SDAs (e.g., SDA OP7) that may concern both design (D) and implementation (I) activities. The prevision was not weighted during the elicitation step because the elicited arguments were not project-specific. They can be reused among projects with other situational factors. The arguments are weighted during the analysis phase where a specific project is analyzed.

IV. ASSET CONSUMPTION PHASE

During the asset consumption phase, we used the factors elicited in the creation phase to engender multi-dimensional views for assessing the impact of factors in different contexts. It is a three step phase. The planning step selects factors and builds generic views of networked arguments related to these factors. The execution step ranks factors according to the specific context of the project and generates weighted views. These contextual views are then used to identify critical factors addressed by designers.

A. Selecting factors and building generic views

The TM pattern was selected as the SDA for analysis. Due to lack of space four activities (M, A, D, I) and five dimensions (F, Q, P, B, S) were considered as factors, and only some of the arguments related to the TM. By selecting activities and dimensions we obtained a generic multi-dimensional view of the TM arguments that relate to the factors under analysis. Table VII presents the view obtained from the arguments described in Table VI.

B. Ranking arguments, activities, and dimensions

We used absolute ranking (H: high, M: medium, L: low and X: not relevant) for prioritizing the factors. As a first step, a work team evaluated how much each activity and dimension was relevant to the project. The weighting of activities and dimensions may be different depending on the project's context and nature. These rankings were used for filtering the arguments that were then further analyzed from the multi-dimensional view of Table VII. In addition, the values of the rankings were used for multiplying the weights of the arguments.

TABLE IV. ISSUES RELATED TO THE TM

SDA	Issue description
Co1	The naming convention is not well defined
Be1	The template method behavior is subject to change
Op1	The deferred steps are not well known
Op8	The hook operations are not well identified
Pr1	The object-oriented paradigm is not well mastered
Pr2	The reusability objectives are not well defined
Pr3	The extensibility objectives are not well defined
St3	The programming language is not well mastered

TABLE V. ARGUMENTS RELATED TO THE TM

Id	Issue or Claim	Rea	Reb	All	Dim	Act
1	A final method cannot be overridden by subclasses		7		+ FQ	DI
6	The low cohesion reduces the analysability of			9	- BPQS	ADI
11	The low cohesion makes maintenance more tedious	5, 6			- BFPQS	ADIM
15	The template method is subject to change	22, 24, 25, 26			- BQS	DI
22	The extensibility objectives are not well				- BQS	ADI
24	There are too many primitive methods				- Q	DI
25	The deferred steps are not well known	22			- BQS	DI
29	The hook operations are not well identified	22			- BFQS	DI

The arguments that relate to the most prioritized activities and dimensions produced more remarkable values in the contextual (i.e., quantified) view. As a second step, the work team estimated how much each argument was relevant to the project. The ranking of the arguments generated the concrete quantified views. As a result, the arguments were then contextualized and their weights calculated. Each argument is potentially the root of an argumentation with reasons, rebuttals, and alleviations. Therefore, the weight of an argumentation is the sum of its rank (H, M, or L) and the ranks of its constituting arguments divided by the number of nodes in the argumentation.

C. Identifying critical factors

Weighting activities, dimensions, and arguments generated contextual views that were used to identify critical factors of the project, which correspond to the cells of views that have remarkable values. The cells were prioritized based on their values. A total impact value was computed for each cell by summing the multiplied weights of the arguments it contains. These values were translated into priorities (1 is the highest priority). Our approach suggested reasoning further about the factors that relate to the most prioritized cell in order to nullify or reduce its value. We made the assumption that taking actions to address these most influent factors produces the greatest benefit.

After the critical factors were addressed, their ranking was adjusted. The adjusted rankings provided new priorities. The user iterates these steps (i.e., identifying flaws and taking actions) until satisfied with the values in the concrete views (i.e., specific threshold values attained).

One of the experiments where the SAM framework was applied was an undergraduate course of object-oriented software design at ETS [11]. The project analyzed in this experiment focused on the design and implementation of a software framework that provided the skeleton of a dice game (DGSF). Table VI presents a contextualization of the factors. Table VII presents a view for the DGSF. In addition, a tool-support was used for managing the SDAs, SSMs, and arguments of the case study. The SDAs manager was developed using the Java programming language and Eclipse development platform. The SSMs and arguments manager was developed using a Java-based compiler and a grammar.

V. Related Work

Many organizations maintain SDAs and AK in a database to assist document control, development, and maintenance activities. Much AK and support for designers provided in the literature includes design decision, design rationale, pattern, tactic and quality model [1, 2, 3, 4, 7, 8, 9, 10]). However, most of these models, methods, and tools provide limited views into the AK database [3, 7, 8]. Many approaches have been proposed to support the design process [2, 3, 7], but few [8] support designers to manage and keep track of the AK. We believe our approach can be used to describe SDAs in a manner that may facilitate selecting relevant AK to keep track of selected SDAs as quantified design decisions. The SAM framework can be used to analyze the AK using structured views that relates in a finer-grained manner the artifacts of the problem space to those of the solution space, from organizational goals to specific system artifacts. We believe a multi-dimensional view is a valuable artifact for providing an integrated view of architectural knowledge.

VI. Conclusions

This paper presented a SAM framework that supports AK management and a multi-dimensional analysis approach to analyzing SDAs such as design patterns and related AK. The proposed AK model describes AK using a set of factors of influence such as SDAs, issues, arguments, activities, and dimensions. Relating these factors enabled the creation of multi-dimensional views that support designers in identifying and addressing critical factors to their projects. The approach was used in industrial and academic contexts. As a proof of concept a prototype tool was developed that students used for analysis. The case studies and controlled experiments produced evidence that the multi-dimensional analysis approach supported by the SAM framework is a valuable step towards handling SDAs and the related AK as an integrated set of factors of influence. The proposed approach may be customized to better support particular SAD processes and system needs. In the near future, it will be supported by an AK management tool. One goal of this work was to contribute to building an AK model of factors linked formally and exploited by algorithms. Finally, five case studies and three controlled experiments have been conducted and will be presented in a forthcoming paper.

TABLE VI. Ranking factors for the DGSF

Ranking of activities for analysis	
Architecting	M
Designing	H
Implementing	M
Managing	L
Ranking of aspects for analysis	
Budget	X
Functions	M
People	M
Quality	H
Schedule	M

Ranking of arguments for each iteration			
Arg.	**Iter1**	**Iter2**	**Iter3**
1	L	L	L
2	H	H	H
6	M	L	X
7	L	X	X
9	H	H	H
11	X	X	X
15	L	L	L
22	H	L	X
24	X	X	X
25	H	X	X
29	X	X	X

TABLE VII. Concrete views of DGSF arguments

Activity	**Aspect**			
Iteration 1	**F**	**P**	**Q**	**S**
A	10	11	3	8
D	6	5	1	2
I	13	12	4	7
M	16	15	9	14
Iteration 2				
A	7	6	1	5
D	14	13	16	12
I	11	10	15	9
M	8	4	2	3

References

[1] Gamma, E., Helm, R., Johnson, R., Vlissides, J., "Design Patterns: Elements of Reusable Object-Oriented Software", A.-W., B. (1995).

[2] Clements, P., Bachmann, F., Bass, L., Garlan, D., Ivers, J., Little, R., Nord, R., Stafford, J., "Documenting Software Architectures – Views and Beyond", Addison Wesley, Boston (2003).

[3] Bass, L., Clements, P., Kazman, R., "Software Architecture in Practice", Addison Wesley, Boston (2003).

[4] Standard, I.: ISO/IEC 42010 Systems and Software Engineering – Recommended Practice for Architectural Description of Software-Intensive Systems. ISO/IEC 42010, (2011).

[5] Carlos, C, Jarred, M, Sanjay, M, Iyad, R, Chris, R, Guillermo, S, Matthew, S, Gerard, V, Steven, W, "Towards an argument interchange format", *Knowl. Eng. Rev.* 21, 4 (Dec. 2006), 293-316.

[6] Wiegers, K.E., "Standing on Principle," *Journal of the Quality Assurance Institute*, vol. 11, no. 3 (July 1997).

[7] Kim, S., Kim, D.K., Lu, L., Park, S., "Quality- driven Architecture Development Using Architectural Tactics", Journal of Systems and Software 82, 1211- 1231 (2009).

[8] Ovaska, E., Evesti, A., Henttonen, K., Palviainen, M., Aho, P., "Knowledge Based Quality-driven Architecture Design and Evaluation", Journal of Info. and Soft. Tech. 52, 577-601 (2010).

[9] Shahin, M., Liang, P., Khayyambashi, M.R., "Architectural Design Decision: Existing Models and Tools", In: WICSA/ECSA 2009, pp. 293-296. IEEE, Cambridge (2009).

[10] Parizi, R.M., Ghani, A., "Architectural Knowledge Sharing (AKS) Approaches: a Survey Research", Journal of Theoretical and Applied Information Technology, 1224--1235 (2008).

[11] Adam, S., El-Boussaidi, G., "A multi-dimensional approach for analyzing software artifacts", 25th SEKE, June 27-29, Boston (2013).

[12] Adam, S., El-Boussaidi, G., Abran, A., "An approach for classifying design artifacts", 27th SEKE, June 6-8, Pittsburgh (2015).