Do Architecture Design Methods Meet Architects' Needs?

Davide Falessi
University of Rome
"Tor Vergata"
DISP
Rome, Italy
falessi@ing.uniroma2.it

Giovanni Cantone
University of Rome
"Tor Vergata"
DISP
Rome, Italy
cantone@uniroma2.it

Philippe Kruchten
University of British Columbia
ECE
Vancouver, Canada
pbk@ece.ubc.ca

Abstract

Several Software Architecture Design Methods (SADM) have been published, reviewed, and compared. But these surveys and comparisons are mostly centered on intrinsic elements of the design method, and they do not compare them from the perspective of the actual needs of software architects. We would like to analyze the completeness of SADM from an architect's point of view. To do so, we define nine categories of software architects' needs, propose an ordinal scale for evaluating the degree to which a given SADM meets the needs, and then apply this to a small set of SADMs. The contribution of the paper is twofold: (i) to provide a different and useful frame of reference for architects to select SADM, and (ii) to suggest SADM areas of improvements. We found two answers to our question: "do architectural design methods meet the needs of the architect?" Yes, all architect's needs are met by one or another SADM, but No, no architectural design method meets simultaneously all the needs of an architect. This approach may lead to improvements of existing SADMs.

1. Introduction

Software architecture (SA) plays an essential role for achieving intellectual control over a sophisticated system's enormous complexity [27]. Software, architects can select among several methods for deriving a software architecture from software requirements; such methods are called "Software Architecture Design Methods" (SADM). There are several surveys describing SADMs, but all of them were done in a comparative way, highlighting similarities and differences, but without dealing with real architects needs; in fact, each SADM (which should be considered as a solution for an architect) was described based on the properties of some other SADMs (i.e., the solution space).

In this paper we will describe SADMs by considering which architects' needs they meet (i.e., the problem space). We first identify and classify the software architects needs in nine coarse categories by searching literature and consulting industrial architects. Then we propose an ordinal scale for each of those needs, and eventually evaluate the completeness of any SADM with respect to those categories of needs. The set of categories of needs that we use may be incomplete (in number) but is relevant to architects.

The contribution of the paper is twofold: (i) to provide a different and useful frame of reference for architects to select SADM, and (ii) to suggest SADM areas of improvements. In other words, we try to help: i) The **architects** in answering the question "Which SADM can help me in addressing a set of my needs?" (rather than "Which SADM is the best for addressing a certain need?") and ii) the **SADM developers** in answering the question "Which set of needs are not yet covered by an SADM?" rather than "How to cover needs not yet covered by any SADM?".

The remainder of this paper is organized as follows. Section 2 introduces to fundamental concepts. Then Section 3 briefly sketches on the rationale of our approach, and Section 4 describes attributes and values of evaluation model that we propose. Sections 5 and 6 present the result, and comments of that evaluation model. Section 7 describes possible future works and concludes the paper.

2. Context and background

2.1 Software architecture and its design

Based on a definition by Shaw & Garlan [41], Kruchten defines SA as "the set of significant decisions about the organization of a software system: selection of the structural elements and their interfaces by which a system is composed, behavior as specified in collaborations among those elements, composition of these structural and behavioral elements into larger subsystem, architectural style that guides this organization. Software architecture also involves usage, functionality, performance, resilience, reuse, comprehensibility, economic and technology constraints and tradeoffs, and aesthetic concerns" [25]. However, despite the fact that software architecture importance is well recognized by the whole software community, there is yet no agreement on the definition of SA.

The existences of SA workshops [38] and the presence of the term "working" in the title of the main conference on software architecture [39] testify the absence of a complete method for designing a SA. Difficulties in designing an architecture include [15]:

- Requirements are frequently captured informally while software architecture is specified in some formal manner. Consequently, there is a semantic gap to deal with.
- Non-functional requirements are difficult to specify in an architectural model.
- Software is often developed in an iterative way. Certain types of requirements cannot be well defined in the early development phases [31]. This implies that sometimes architects take their based upon vague requirements.
- Stakeholders dealing with software architecture issues view the system from different perspectives. This usually implies conflicting goals, expectations, and used terminology.

However, there are three main sources of architecture [24]:

- **Reuse**: architecting is a difficult task; the fact that a system was successful and it used a certain component, rule or responsibility demands the reuse of such issues. Source of reuse can be the earlier version of the system, another system sharing key characteristics, architectural pattern [1]. Domain-specific Software Architecture [46] and Product Line Architecture [6] are different examples of two perfect marriages between software architecture and reuse.
- **Method**: language, process model, and a systematic and conscious related technique for bridging the gap between software architecture and requirements. The present paper deals with such an issue.
- **Intuition**: invention of software architecture elements based on experience.

The ratio among the 3 software architecture sources is dependent on the experience of architects and the novelty of the domain.

In general, every approach aimed to solve a problem enacts an iterative process consisting in the following three phases (see Figure 1):

1. **Understand the problem**: Such phase consists in analyzing the problem and extract the real needs from the ambiguous and huge problem description.
2. **Solve the problem**: Provide a solution as best as possible by analyzing the abovementioned needs and the characteristics of known solutions.
3. **Evaluate the solution**: Decide if the provided solution solves the problem.

From phase 3, the process re-enters phases 1 or 2 when the provided solution is not acceptable but the problem is still considered feasible. The process ends when the provided solution is considered acceptable or the problem is evaluated as unfeasible.

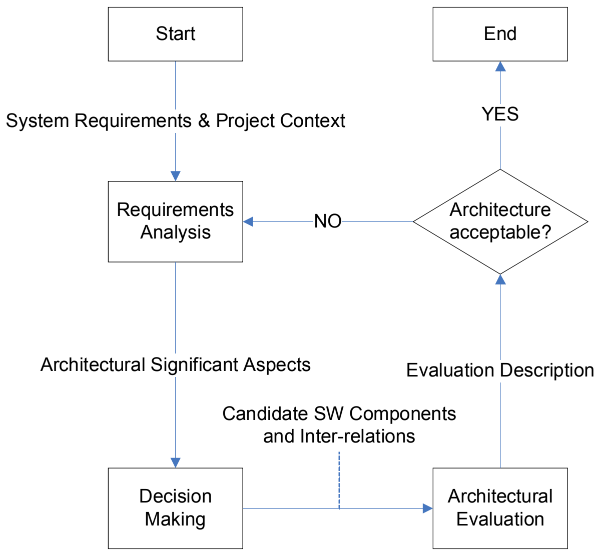

Figure 1: Problem solving activities and architectural design activities.

In the SADM's context, the problem consists in the system requirements, and the solution consists in the characteristics of the software components and their inter-relations. Despite the core of SADM is intrinsically the phase number 2, experience shows that such phase, in order to be enacted successfully, should be carefully integrated in the above-mentioned process [17]. Fowler [13] and Roshandel [35] confirm this structure by arguing the importance in SADM of the phases 1 and 3 respectively.

Despite the fact that "many of the design methods were developed independently, their descriptions use different vocabulary and appear quite different from each other," … "they have a lot in common at the conceptual level." [17]. Differences among SADMs include the level of granularity of the decision to make, concepts taken into account, level of emphasis on phases, audience (large vs. small organization) and usage domain. A sound discussion regarding SADM variability and commonalities is out of the scope of the present work and can be found in [17].

2.2 Survey of software architecture design methods

2.2.1 Approaches and techniques

From the available approaches to comparing design methods, we adopted the quasi-formal approach that Sol suggested [42]. Based on Song & Osterweil [44], such a comparison can be enacted by five different techniques:

1. Describe an idealized design method and evaluate other methods against it.
2. Distill a set of important features inductively from several methods and compare those methods against it.
3. Formulate a priori hypotheses about a method's requirements and derive a framework from the empirical evidence in several methods.
4. Define a meta-language as a communication vehicle and a frame of reference against which you can describe many methods
5. Use a specific contingency approach and try to relate each method's feature to specific problem.

As briefly described in the remaining of this sub-section, the literature is limited to present SADM comparisons of using technique one, two, and four, in our best knowledge. The approach taken in the present paper uses the third of the comparison techniques above, and extends it by taking into account industrial architects' needs, as better detailed in Section 3.

2.2.2 Surveys

Hofmeister *et al.* [17] compared five industrial software architecture design methods and extracted a general software architecture design approach from commonalities of those methods. Such a design approach inspired the one that we already presented (see Figure 1). The work by Hofmeister *et al.* also provided a solid foundation to this paper.

Bahill *et al.* [1] provided and applied to eleven design methods a "benchmark" application for comparing design methods. Sharble & Cohen [40] compared class diagrams from two different OO development methods by using complexity metrics. Wieringa [49] compared artifacts from structured and OO specification methods. Hong *et al.* [18] compared artifacts from six OO analysis and design methods. Kim & Lerch [22] quantified the cognitive activities of designers when using an OO design method and a functional decomposition method, respectively. Song & Osterweil [43] proposed and used a process-modeling technique to model activities and artifacts from some methodologies (Jackson System Development, Booch's Object Oriented Design, Structured Design, and Rational Design Methodology).

Hong *et al.* [18] first analyzed the available methods by considering their activities, breaking down these activities by using a finer granularity, and selecting the best sub-activities; then they created a "super-methodology" by combining those sub-activities; eventually, they compared their super-methodology with each of the methods they had been decomposing. Similarly to Hong *et al*., Fichman & Kemerer [12] compared design methods by taking the superset of activities supported by the methods they were analyzing. Eventually, they used eleven analysis activities and ten design activities. Dobrica & Niemela [7] compared eight methods for software architecture evaluation (see phase 3 of the process model in Section 2.1).

2.3 Software architecture design methods

There is not enough room here for comprehensive description of SADMs. Of course, there are already in literature papers and surveys that concern SADM; in order to help the reader to approach our SADM context, Table 1 briefly sketches some SADMs key aspects, such as references and specificities.

3. Rationale of our Approach

As we mentioned in Section (point 3), in order to describe and analyze completeness of an SADM with respect to software architects needs, we started by analyzing the literature and consulting professional software architects. In fact, this step was aimed to understand the problem space of architecting software. Subsequently, we proceeded to synthesize that problem space in nine categories of needs (called here: coarse-grained needs). Such categories of needs constitute the attributes of an SADM evaluation model. This might neglect some software architect needs, however it certainly takes in consideration real needs.

The specificity of our SADM evaluation model is its total independence from the SADM state of the art. As a matter of fact, our evaluation model originates from the architects' needs rather than from SADM comparative analyses; this is what makes the present description new and original.

Our description of SADMs is based on handling them as available solutions for software architects for the problem of designing a SA. Intuitively, a description of a solution based on the description of the problems it is able to solve is understandable because it is direct. Moreover it is a useful means to strongly separate the problem space (i.e., evaluation model) from the solution space (i.e., evaluation model results). In other words, the result of this model is related to the current SADM state of the art.

Let us highlight that the aim of our model is not to express qualitative evaluations about actual SADMs but to characterize them from the point of view of a software architect.

Nine categories of needs compose our evaluation model, each category matching a precise type of software architect need. The categories of needs are orthogonal one to another.

In order to enact a measurement model and improve the understandability of the measurements resulting from the application of the evaluation model to SADMs, we design three specific needs for each category of need (called here: fine-grained needs), and measure the fulfillment of such a need by a four points ordinal scale. In particular, whatever the current category of architects need might be, an SADM is able to fulfill that need at one of the following levels:

- **Null** (N): the evaluated method does not include a mandatory element.
- **Minimum** (Mn): specifies a minimum quantity of an element that a SADM is requested to fulfill (else the evaluated method is not considered as an SADM) or, in case, an optional element. In the former, the Mn specification is given positively ("the method does include …"); in the latter, it might be given negatively ("the method does not include …"), i.e. Mn≡Null and the method is still a candidate SADT (for the current category of need).
- M**aximum** (Mx): specifies the best fulfillment that a SADM could reach (for the current category of need).
- **Intermediate** (I): represents fulfillments other than Mn, Mx, and N.

N, Mn, I, and Mx are values of an ordinal scale; usually, N<Mn<I<Mx; sometimes, N≡Mn<I<Mx.

In conclusion, in order to evaluate an SADM we provide and utilize a structure of direct measurement models, whose components are the category of needs, rather than a synthesis of that structure in an indirect model; a four points ordinal scale (N, Mn, I, Mx) measures directly the fulfillment of every category of need. Limits and threats to validity are described in Section 6.1 for such a structure of measures.

4. Architects Needs

In this section, for each category of needs, we describe:

I. Key characteristics and why it is relevant for software architects. To these aims, we reference the literature and synthesize on what software architects told us.

II. Specific needs or levels, from Null up to Maximum.

Table 1: **Key aspects of SADM (as claimed by the authors of the method) and our comments**

SADM	References	Peculiarities & *Our comments*
G&S	[29]	Goal & Scenario consists in combining goals and scenario mechanisms. Include design rationale documentation. *Still promising.*
Tropos	[14]	Tropos is suitable for agent-based, cooperative, and distributed systems; it provides support for organizzation issues; it does not provide guidelines but ad hoc process.
Rule Based	[28]	Rule Based adopts a unified language for describing both SA and requirements. It is rule-based. *Still promising.*
RUP	[13], [17]	The Rational Unified Process well integrates SADM in the whole sw. development process, has high level of standadirzation, is driven by risk mitigation. *It should be difficult top use RUP SADM in other sw development process.*
Siemens 4V	[16], [17]	Based on their Global Analysis phase, which aims to analize and drive iterations; the architect identifies and analyzes the factors, explores the key architectural issues or challenges, then develops design strategies for solving these issues. *Developed a*
BAPO	[1], [17]	Business Architecture Process and Organization supports the development of product families, is flexible, adaptive, and goal-oriented, provides support for short innovation cycles, is suitable for embedded systems
ASC	[33], [17]	Architectural Separation of Concerns is based on goals, architectural decisions, architecture descriptions, verification of architectural decisions, consistency-check of architecture and implementation. *Developed and used industry-wide* .
CBSP	[15]	Component-Bus-System, and Properties provides a model to locate as a layer between requirements and SA. *High level of formal documentation.*
ADD	[2], [17]	Attribute Driven Design is the only non-functional requirements-driven method. *Provides several well-assembled SADM activities. Not yet standardized.*

4.1 Abstraction and refinement

I. In SADM, we can look at refinement and abstraction as kinds of complementary concepts: the former adds details (top-down direction), the latter removes details (bottom-up direction) from software architecture model(s). The Object Management Group [32] has been promoting the usage of abstraction and refinement in SA design: the Model Driven Architecture emphasized on such concepts [30].

II. Levels:
- N≡Mn:: SADM does not include guidelines either for refinement or abstraction activity.
- I:: SADM includes guidelines only for refinement or only for abstraction activity.
- Mx:: SADM includes guidelines for both refinement and abstraction activity.

4.2 Empirical validation

I. Software engineering is human-based; consequently, empirical investigations can strongly help in evaluating its products, processes and tools. The relevance of empirical software engineering is demonstrated by the presence of specific international conferences [45], journals [9], books [48], and research groups [34].

II. Levels:
- N:: Method still in the stage of not yet verified proposal.
- Mn:: SADM proposed and verified on paper only.
- I:: SADM already used by professionals or experimentally validated in lab.
- Mx:: SADM used both by professionals and experimentally validated, or extensively used by professionals.

4.3 Risk management

I. The perspective taken in this work is that risk is inherent to any software development activity [1]. The inevitability of risks demands for guidelines to recognize and manage risks.

II. Levels:
- N≡Mn:: SADM does not take into account risk concepts.
- I:: SADM provides guidelines to handle different categories or levels of risk.
- Mx:: SADM provides guidelines to handle different categories and levels of risk.

4.4 Interaction management

I. Techniques to address non-functional requirements interact to each others [8]; therefore, the effects of the adoption of a technique depend on other techniques: the ones that are already in use or being employed [23]. In such a context, in our view, techniques include architectural styles, patterns, and mechanisms.

II. Levels concerning Interaction Management category of need are:
- N≡Mn:: SADM does not provide guideline of any category for handling interactions among techniques and managing related issues.
- I:: SADM provides generic guidelines for handling interactions among techniques and managing related issues.
- Mx:: SADM provides specific guidelines for handling interactions among techniques and managing related issues.

4.5 Tool support

I. Complex decisions demand for tool support. The multiplicity of the relation between software architecture and requirements is many to many, i.e. an architectural decision can affect several requirements, and vice versa; the human intelligence can be in trouble while architecting, when dealing with a huge number of requirements (i.e. large software system) [15]. Nowadays, the importance of the automated support in software engineering is confirmed by the presence of specific international conferences (both the automated software engineering [20] and the newer software engineering decision support [36]), journals, books, and research groups [37].

II. Levels:
- N≡Mn:: there is no tool available to support SADM.
- I:: SADM is supported only by passive tools, i.e. the tools available provide only the “visualization” feature (i.e. graphs, matrices, etc.)
- Mx:: SADM is supported by active tool, i.e. the tools available include passive tools and additionally provide features to realize inference (simulation, worst/wrong solution avoidance, consistency checking).

4.6 Concerns

I. In general, views help architects to keep separated the different concerns of interests. The software community as a whole seems to agree that using multiple coordinated views to describe an SA provides high benefits. Several standards already exist, each proposing a different of basic views [5]; a popular one (4+1 views) was proposed by Kruchten in [26].

II. In order to allow an evaluation, let us simplify the domain of the SA “concerns” by assuming the spectrum of such concerns as: “Functional”, “Behavioural” and “External” (or “Physical”). Related levels are:
- N:: SADM deals with no one of the concerns above.
- Mn:: SADM deals with one concern.
- I:: SADM deals with two concerns.
- Mx:: SADM deals with all (three) the concerns above.

4.7 Knowledge base

I. This category is the architect's need to be supported in his decision making by some form of knowledge contained in a tool adapted to the method

II. Levels:
- N≡Mn:: SADM related tool does not use any experience base for assisting architects in decision making.
- I:: SADM related tool use an experience base for assisting architects in decision making.

Such experience base is static: it cannot be changed (i.e. improved) during its usage.
- Mx:: SADM related tool use an experience base for assisting architects in decision making. Such experience base is dynamic: it can be changed (i.e. improved) during its usage.

4.8 Requirements change management

I. The ability in reacting to requirements changes is crucial to achieve project success. Such a feature is mostly used in the beginning of a software development process, when requirements are still incomplete and not well defined (see Section 2.1).
II. Levels:
   - N≡Mn:: Views and requirements are not traced.
   - I:: Guidelines are provided to trace requirements from one view.
   - Mx:: Guidelines are provided to trace requirements from many views.

4.9 Number of activities

I. The main classes of activities in a SADM are: "Requirement analysis", "Decision making", and "Architectural evaluation" (see Section 2 above); these provide equally important contributes for arriving to a SA. A method dealing with one of them has is a method for just that activity, not a SADM; in our view, only methods which provide guidance for enacting two or all those activities are SADM.
II. Levels:
   - N:: the SADM provides guidelines for enacting no more than one SADM activity.
   - Mn:: SADM provides guidelines to enact two SADM activities.
   - Mx:: SADM provides guidelines to all the SADM activities.

5. Architects needs and methods not taken into consideration

Despite the management of SA evolution is one of the main challenging aspects in practice [21], we remand its analysis to a future and more focused work.

We did not consider Problem Frame [3] and Softgoal [4] SADMs because they are mainly included in CBSP and G&S, respectively.

Despite empirically validations [11] provided evidence that architectural design rationale documentation is useful [47], this is not yet used largely in industry [10]. In fact, it seems to us that G&S is the only SADM to include its own guidelines for documenting explicitly design decision rationale. Anyway, we postpone to future works the investigation of what SADM includes what design documentation rationale.

6. Results and comments

6.1 Introduction, limits and threats to validity

We do not take into account quality issues in SADM: in our view, such a topic should be analyzed empirically rather than theoretically in order to provide useful results in the short term. This point is postponed to further ongoing research.

In case where an SADM should not meet a specific architect need, we can conjecture to utilize that SADM in combination with an additional method, which is aimed to meet that need. However, in order to support the combination of an SADM with additional foreign methods, specific guidelines and empirical evidence should be provided.

Concerning all the SADM that this paper is considering, we contacted their authors and encouraged them to rate their own methods by using our evaluation model. Eight of them kindly complied. Because these evaluations are consistent with our own, we will be using them in the remaining of this paper; for the rest of the SADMs, we will be using our evaluations, based on what we know about, as reported by literature (see column References in Table 1).

Threats to validity of those evaluations mainly come from the attributes of the models that depend on the evaluator subjectivity, including:

- Using SADM out-of-date documentation
- Misunderstanding the documentation.
- Evaluating as not provided by the SADM the features that the documentation neglected.
- Overestimating, when the SADM authors also performed in the role of evaluators.
- Misunderstanding the proposed category of needs and the related specific needs, when the SADM authors also performed in the role of evaluators

In order to gain on validity, we tried to mitigate the impact of subjectivity on evaluations by enacting several precautions, including:

- Collecting advices from several architects, who work in different contexts, and using those advises as drivers for selecting the categories of needs to put in our evaluation model.
- Avoiding ambiguity by using both a standard terminology [19], and a formal and standard ordinal scale for measuring the model attributes.
- Employing one more evaluator, when the first evaluator was the author of the method, and verifying the consistency of the results.

Concerning the last point above, let us note explicitly that it also gives our measurement model a first accredit. In fact, almost the totality of measures resulted consistent, which different subjects—the SADM authors and ourselves—independently provided for common objects.

6.2 Reference for helping an architect to select the SADM mostly appropriate for the needs

Table 2 provides an overview on the SADM levels of fulfillment for architects needs. We are convinced that Table 2 constitutes a valuable needs-driven support for architects who are looking for a SADM.

All the results described in the present section derive from Table 2.

Figure 2 shows the degree of completeness of SADMs with respect to the categories of needs. Figure 2 constitutes an overall picture of the state of the art in the SADM domain, rather than a further reference for architects. In fact, as already mentioned, this paper (i) did not synthesize on the attribute of our evaluation model, and (ii) neglected quality issues. As a consequence, we are not (yet) able to provide a measurement model able to answer to questions like: What is the best SADM for these specified needs and quality? and: What is the accuracy of that result?

6.3 Areas of improvement in SADMs

6.3.1 Meeting one category of needs at a time

Figure 3 presents what percentage of SADMs meets a level of architects needs. Based on Figure 3, we can deduce that SDAM developers assign greater importance to features like "Requirements Change management" and "Concerns", and lesser importance to features like "Knowledge Base Support" and "Empirical Validation".

Table 2: Architects Needs Model Results.

	Activities	Empirical Validation	Knowledge Base Support	Refinement and Abstraction	Requirements Change Mng.	Risk Management	Solution Interaction Mng.	Tool Support	Concerns
ADD	Mx	Mi	I	Mx	Mx	Mi	Mx	Mi	Mx
ASC	Mx	I	Mi	I	Mx	Mx	I	Mi	I
CAFCR	Mi	Mx	Mi	Mx	Mx	Mx	I	I	Mx
CBSP	Mx	I	Mx	Mx	Mx	Mi	I	Mx	Mx
G&S	Mx	I	I	Mx	Mx	I	I	Mx	I
Rule Based	Mi	Mi	Mx	Mi	Mx	Mi	Mx	Mx	Mi
RUP	Mx	Mx	Mi	Mx	I	Mx	I	Mx	Mx
Siemens 4v	Mi	Mx	Mi	I	Mx	Mx	Mx	I	Mx
Tropos	Mi	Mi	Mi	Mi	Mx	Mi	Mx	I	Mx

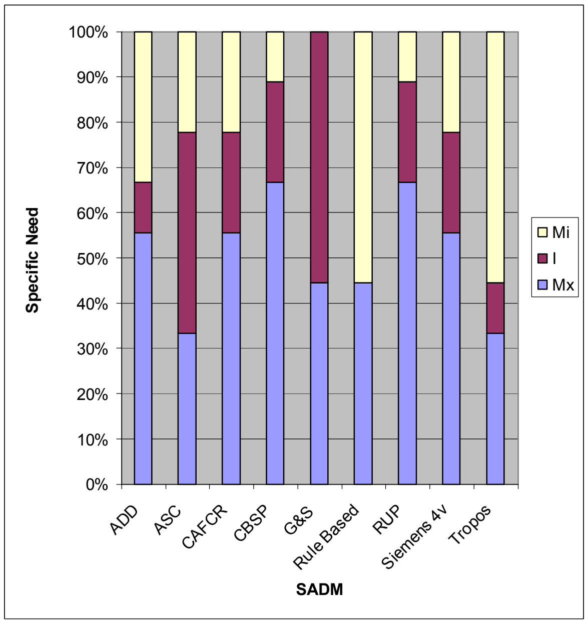

Figure 2: Software Architecture Design Methods with respect to specific needs fulfillment level (%).

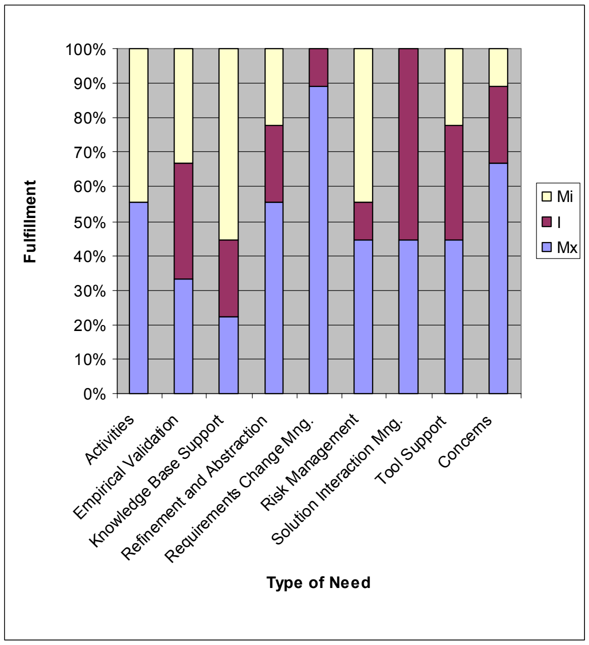

Figure 3: Specific needs fulfillment level (%).

Table 3: Analysis of pairs of categories of needs.

	Activities	Empirical Validation	Knowledge Base Support	Refinement and Abstraction	Requirements Change Mng.	Risk Mng.	Solution Interaction Mng.	Tool Support	Concerns
Activities	6	2	1	5	5	3	1	3	4
Empirical Validation	2	3	0	2	2	3	1	1	3
Knowledge Base Support	1	0	2	1	2	0	1	2	1
Refinement and Abstraction	5	2	1	5	4	2	1	3	4
Requirements Change Mng.	5	2	2	4	8	3	4	3	5
Risk Management	3	3	0	2	3	4	1	1	3
Solution Interaction Mng.	1	1	1	1	4	1	4	1	3
Tool Support	3	1	2	3	3	1	1	4	2
Concerns	4	3	1	4	5	3	3	2	6

Table 4: Pair of categories of needs that no SADM is able to meet.

ID	Needs	
1	Empirical Validation	Knowledge Base Support
2	Knowledge Base Support	Risk Mng.

6.3.2 Meeting pairs of categories of needs

Table 3 shows the number of SADMs that are able to meet a couple of needs. Based on Table 3, we can observe, for instance: five SADMs are able to meet the needs of "Requirements Change Management" and "Activities"; vice versa, all the SADMs considered are unable to meet "Knowledge Base Support" and "Risk management". In other words, architects would be in trouble, if having both those neglected needs. Consequently, SADM developers should concentrate in providing those features. To identify potential SADM improvement areas, Table 4 describes pair of needs unmet by any SADM.

6.3.3 Meeting more than two categories of needs

Table 5 describes how many, and in what percentage, single needs, couple of needs, and so on up to nine-tuples of needs are there, that one or more SADM are able (resp. unable) to meet. Based on Table 5, we can observe, for instance, the considered set of SADMs are able to meet 68% of need-triples (see the item located in the third row and second column of Table 5).

Again, with the objective of identifying further SADM improvement areas, Table 6 describes SADM-unmet triplets of needs. Based on Table 6, we can observe, for example, that there is no SADM, which is able to meet the triple made by needs "Activities", "Risk Management", and "Solution Interaction Management" (see row 6 in Table 6).

7. Conclusion and future work

This paper tried to answer to following question: "Do actual software architecture design methods meet architects needs?" To do so, we define nine categories of software architects' needs, propose an ordinal scale for evaluating the degree to which a given SADM meets the needs, and then apply this to a small set of SADMs.

Based on results from the present study, we argue that there are two answers to this question: a) Yes, they do. In fact, we showed that one or more SADMs are able to meet each individual architect needs that we considered (see row 1, column 2 in Table 5). b) No, they do not. In fact, we showed that there is no SADM which is able to meet any tuple of seven or more needs (see row 7, column 2 in Table 5), which means that there is still some work to do to improve SADMs to actually help architects.

In order to provide directions for SADM improvement, we presented couples of needs (see Table 4), and triplets of needs (see Table 6) that actual SADMs are unable to meet. We hope should provide some valuable input to SADM developers.

Further investigation is needed to better understand *why* the coverage of our need is different across SADMs, and also to extend the list of needs categories.

Table 5: Analysis of architects needs available combination.

# Needs	Fulfilled		Not Fulfilled	
	#	%	#	%
One	9	100	0	0
Two	34	94	2	6
Tree	57	68	27	32
Four	47	37	79	63
Five	19	15	107	85
Six	3	4	81	96
Seven	0	0	36	100
Eight	0	0	9	100
Nine	0	0	1	100

Table 6: Triplets of categories of needs that no SADM is able to meet.

Needs		
Activities	Empirical Validation	Knowledge Base Support
Activities	Empirical Validation	Requirements Change Mng.
Activities	Empirical Validation	Solution Interaction Mng.
Activities	Knowledge Base Support	Risk Mng.
Activities	Knowledge Base Support	Solution Interaction Mng.
Activities	Risk Mng.	Solution Interaction Mng.
Activities	Solution Interaction Mng.	Tool Support
Empirical Validation	Knowledge Base Support	Refinement and Abstraction
Empirical Validation	Knowledge Base Support	Requirements Change Mng.
Empirical Validation	Knowledge Base Support	Risk Mng.
Empirical Validation	Knowledge Base Support	Solution Interaction Mng.
Empirical Validation	Knowledge Base Support	Tool Support
Empirical Validation	Knowledge Base Support	Concerns
Empirical Validation	Refinement and Abstraction	Solution Interaction Mng.
Empirical Validation	Requirements Change Mng.	Tool Support
Empirical Validation	Solution Interaction Mng.	Tool Support
Knowledge Base Support	Refinement and Abstraction	Risk Mng.
Knowledge Base Support	Refinement and Abstraction	Solution Interaction Mng.
Knowledge Base Support	Requirements Change Mng.	Risk Mng.
Knowledge Base Support	Risk Mng.	Solution Interaction Mng.
Knowledge Base Support	Risk Mng.	Tool Support
Knowledge Base Support	Risk Mng.	Concerns
Knowledge Base Support	Solution Interaction Mng.	Concerns
Refinement and Abstraction	Risk Mng.	Solution Interaction Mng.
Refinement and Abstraction	Solution Interaction Mng.	Tool Support
Requirements Change Mng.	Risk Mng.	Tool Support
Risk Mng.	Solution Interaction Mng.	Tool Support

8. Acknowlegements

We would like to acknowledge Patricia Lago for her useful suggestions and the following SADM authors for having been rated their own methods: Len Bass, Paolo Giorgini, Alexander Ran, Wendy Liu, Alexander Egyed, Henk Obbink, and Eric Yu.

9. References

[1] P. America, H. Obbink, & E. Rommes, "Multi-view variation modeling for scenario analysis", Proceedings of Fifth International Workshop on Product Family Engineering 2005, Springer-Verlag, pp. 44–65.

[2] L. Bass, P. Clements, R. Kazman, Software Architecture in Practice, 2nd ed. Addison-Wesley, Reading, MA, 2003.

[3] C. Choppy & G. Reggio, "Using UML for Problem Frame Oriented Software Development". 13th International Conference on Intelligent & Adaptive Systems and Software Engineering, 2004, pp 239-244

[4] L. Chung, D. Gross & E. Yu, "Architectural Design to Meet Stakeholder Requirements", First Working IFIP Conference on Software Architecture, 1999.

[5] P. Clements, F. Bachmann, L. Bass, D. Garlan, J. Ivers, R. Little, R. Nord, J. Stafford, "Documenting Software Architectures: Views and Beyond", Pearson Education, 2003.

[6] P. Clements, L. Northrop, "Software Product Lines: Practices and Patterns", Addison-Wesley, 2001.

[7] L. Dobrica & E. Niemela, "A survey on software architecture analysis methods," IEEE Transactions on Software Engineering, 28 (7), 2002, pp. 638-653.

[8] H. Eguiluz & M. Barbacci "Interactions among techniques addressing quality attributes", CMU/SEI-2003-TR-003.

[9] Empirical Software Engineering, An International Journal, V.R. Basili and L.C. Briand (eds) Springer .

[10] D. Falessi, M. Becker, & G. Cantone, "Design Decision Rationale: Experiences and Steps Ahead Towards Systematic Use", Workshop on SHAring and Reusing architectural Knowledge at the 9th International Conference on Software Reuse, 2006.

[11] D. Falessi, G. Cantone, M. Becker, "Documenting Design Decision Rationale to Improve Individual and Team Design Decision Making: An Experimental Evaluation", ISESE 2006.

[12] R.G. Fichman and C.F. Kemerer, "Object-oriented and conventional analysis and design methodologies," IEEE Computer, 25 (10), 1992, pp. 22-39.

[13] M. Fowler, "Analysis Patterns: Reusable Object Models", Addison Wesley, 1997.

[14] P. Giorgini, J. Mylopoulos, & M. Pistore, "The Tropos methodology: an overview", in: F. Bergenti, M.-P. Gleizes, F. Zambonelli (Eds.), Methodologies and Software Engineering for Agent Systems: The Agent-Oriented Software Engineering Handbook, Kluwer Academic Press, Boston, MA; London, 2004, p. 505

[15] P. Gruenbacher, A. Egyed, & N. Medvidovic "Reconciling Software Requirements and Architectures with Intermediate Models". Journal on Software and System Modeling, December 2003.

[16] C. Hofmeister, R. Nord & D. Soni, "Applied Software Architecture", Addison-Wesley, Boston 1999.

[17] C. Hofmeister, P. Kruchten, R. Nord, H. Obbink, A. Ran & P. America, "Generalizing a Model of Software Architecture Design from Five Industrial Approaches", WICSA 2005.

[18] S. Hong, G. van den Goor, & S. Brinkkemper, "A formal approach to the comparison of object-oriented analysis and design methodologies," in Proceedings of 26th Hawaii International Conference on System Sciences, Wailea, HI, USA, 1993, pp. iv 689-698.

[19] IEEE 1471:2000, "Recommended practice for architectural description of software intensive systems", Los Alamitos, CA: IEEE, 2000.

[20] 21st IEEE/ACM International Conference on Automated Software Engineering, (ASE 2006), http://www.ase-conference.jp.

[21] A. Jansen & J. Bosch, "Evaluation of Tool Support for Architectural Evolution", International Conference on Automated Software Engineering 2004, pp. 375-378.

[22] J. Kim & F. Lerch, "Towards a model of cognitive process in logical design: comparing object-oriented and traditional functional decomposition software methodologies", Human Factors in Computing Systems CHI Proceedings, May 1992, pp.489-498.

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

[24] P. Kruchten. "Mommy, Where Do Software Architectures Come from?" 1st International Workshop on Architectures for Software Systems, Seattle, WA, April 1995.

[25] P. Kruchten, "The Rational Unified Process: An Introduction", 3rd ed. Addison-Wesley-Longman, 2003.

[26] P. Kruchten, "The 4+1 View Model of Architecture," IEEE Software 12 (6), 1995, pp. 42–50.

[27] P. Kruchten, H. Obbink, & J. Stafford, "The Past, Present and Future of Software Architecture", IEEE Software, March/April 2006.

[28] W. Liu, S. Easterbrook, "Eliciting Architectural Decisions from Requirements using a Rule-based Framework", STRAW'03 located at ICSE 2003

[29] L. Liu & E. Yu, "From Requirements to Architectural Design Using Goals and Scenarios". In: Proceedings of the ICSE-2001 (STRAW 2001). Toronto, Canada.

[30] Model Driven Architecture Guide, http://www.omg.org/docs/omg/03-06-01.pdf

[31] B. Nuseibeh, Weaving Together Requirements and Architectures. IEEE Computer 34, (3): 115–117. 2001

[32] Object Management Group web page, http://www.omg.org/

[33] A. Ran, "ARES Conceptual Framework for Software Architecture" In: Jazayeri, M., Ran, A., van der Linden, F. (Eds.), Software Architecture for Product Families Principles and Practice, Addison-Wesley, pp.1–29

[34] D. Rombach, The Fraunhofer Institute of Experimental Software Engineering, http://www.iese.fhg.de/

[35] R. Roshandel, B. Schmerl, N. Medvidovic, D. Garlan, and D. Zhang, "Understanding Tradeoffs among Different Architectural Modeling Approaches," WICSA 2004, Oslo, Norway, 2004.

[36] G. Ruhe, Software Engineering Decision Support, Special Issue International Journal of Software Engineering and Knowledge Engineering, 13 (5), Oct 2003.

[37] G. Ruhe, Software Engineering Decision Support Laboratory, from http://www.seng-decisionsupport.ucalgary.ca/research.htm

[38] 2nd International Workshop From SofTware Requirements to Architectures, STRAW'03, at the ICSE 2003, http://se.uwaterloo.ca/~straw03/, last access: 25-05-206.

[39] Sixth Working IEEE/IFIP Conference on Software Architecture, WICSA 2007, http://www.gv.psu.edu/WICSA2007/, last accessed: 25-05-206.

[40] R. Sharble R. & S. Cohen, "The object-oriented brewery: a comparison of two object-oriented development methods," ACM SIGSOFT Software Engineering Notes, 18 (2), 1993, pp. 60-73.

[41] M. Shaw & D. Garlan. "Software Architecture. Perspectives on an Emerging Discipline." Prentice Hall, 1996.

[42] H. Sol, "A Feature Analysis of Information Systems Design Methodologies Through Process Modelling," Proc. 1st Int'l Conf. software Process, IEEE CS Press, Los Alamitos, 1991.

[43] X. Song & L. Osterweil, "Experience with an approach to comparing software design methodologies," IEEE Transactions on Software Engineering, 20, (5), 1994, pp. 364-384.

[44] X. Song, & L. Osterweil, "Toward Objective, Systematic Design-Method Comparisons", IEEE Software, 9 (3), pp. 43-53, 1992.

[45] The 2006 Experimental Software Engineering International Week (ESEIW 2006), http://www.cos.ufrj.br/~ght/eseiw2006.htm

[46] W. Tracz, DSSA (Domain-Specific Software Architecture) Pedagogical Example. ACM SIGSOFT Software Engineering Notes, 1995

[47] J. Tyree & A. Akerman, "Architecture Decisions: Demystifying Architecture", IEEE Software, 22, (2), Mar/Apr, 2005.

[48] C. Wohlin, P. Runeson, M. Höst, M.C. Ohlsson, B. Regnell, & A. Wesslén: "Experimentation in Software Engineering: An Introduction", The Kluwer International Series in Software Engineering, 2000.

[49] R. Wieringa, "A Survey of Structured and Object-Oriented Software Specification Methods and Techniques," ACM Computing Surveys, vol. 30, no. 4, 1998, pp. 459-527.