

An Analysis of Decision-centric Architectural Design Approaches

Wanfeng Bu, Antony Tang, Jun Han

Swinburne University of Technology, Australia

wbu@swin.edu.au, atang@swin.edu.au, jhan@swin.edu.au

Abstract

Emerging research suggests that software architecture can be represented as a set of design decisions. Several decision-centric architectural approaches have been proposed, which provide methodological support to design reasoning and justification. However, these approaches have different premises and they address different aspects of design reasoning. We analyze this developing field from three perspectives: architectural knowledge modeling, architectural design techniques and design rationale capture and use. From this analysis we have identified areas of improvements such as ASR identification, design reasoning process and architectural knowledge management.

1. Introduction

Software architecture serves as the blueprint of software systems. The IEEE 1471-2000 standard [1] defines software architecture as the fundamental organization of a system embodied in its components, their relationships to each other and to the environment and the principles guiding its design and evolution. However, most architectural designs only depict components and connectors, but omit the design decisions and design rationale. This may lead to 1) costly support for system evolution 2) lack of stakeholder communication and 3) limited reusability of software system [2].

To address these problems, new approaches are developed in recent years that support design reasoning and document design decisions as core entities of software architecture. These approaches have some common features: 1) treat architecture design as a design decision process; 2) manage architectural knowledge, emphasizing the explicit organization and documentation of design decisions; 3) integrate techniques/methodologies in the design reasoning process; 4) capture and document design rationale. In

this paper, we analyze nine methods: (1) Archium [3, 4]; (2) Akerman and Tyree's Ontology (ATO) [5]; (3) AREL [6]; (4) ArchDesigner [7]; (5) Bayesian Belief Network based Alternatives Selection Method (BBNbASM) [8]; (6) AQUA [9]; (7) Automated Solution Synthesis Method (ASSM) [10]; (8) PAKME [11]; (9) ADDSS [12].

While these software architecture decision-centric approaches share some common characteristics, the architectural knowledge they record and the design reasoning techniques they employ differ greatly, and as a result the reasoning process and the resulting design rationale vary. The main objectives of this study are twofold. Firstly, it aims to provide a bird's-eye view of this developing field, and to identify the differences and the similarities of these approaches. Secondly, through the analysis, we aim to find the strengths and weaknesses of these approaches to uncover new research alignments and questions.

In this paper, we use three key perspectives to analyze the nine decision-centric methods. These perspectives are based on the referential models on software architecture knowledge management and design rationale. We apply these methods to an industry case study to compare the decisions and the design outcomes that each one of them produces. We have found that future improvements in decision-centric approach can be made by identifying architecturally significant requirements (ASRs), improving current design reasoning processes and extending architectural knowledge management to support the software life cycle.

2. Analysis framework

Different approaches have made different assertions, mostly implicitly, on what design reasoning is fundamentally about. For instance, the decision issues addressed by the methods vary from low-level software design decisions to high-level managerial decisions, and from disparate decisions to interrelated decisions.

Even though they have a common theme of applying some design decision making mechanisms, there are apparent differences in many areas. We found our comparisons on three key software architecture perspectives: software architecture knowledge, decision making techniques and design rationale management. The perspective of software architecture knowledge model provides a baseline on the information that can be captured and used in decision-centric approaches. The perspective of decision making techniques deals with the process with which design decisions are made. The perspective of design rationale management deals with the techniques for capturing and reusing design rationale to support software architectural design.

Architectural knowledge supports software engineering. The IEEE 1471-2000 standard suggests a conceptual framework for architectural description but its reference to design decisions and design reasoning is limited. Kruchten's ontology [13] and Tyree's Decision Template [14] compensate this deficiency by emphasizing design decisions and design rationale. A more recent definition is provided by an architectural knowledge model, i.e. the Core model [15]. It describes the key elements of architectural knowledge and design reasoning with minimalistic but complete concepts [15]. Since these standards and reference models complement each other, we refer to them as the baselines for the analysis of decision-centric approaches.

Perspective 1: Decision-centric architectural knowledge modeling

The elements in the Core model can be grouped in terms of the conceptual domains that they deal in: problem space, decision space and solution space. In the problem space, the key concepts are *design concerns* and *decision topics*. Stakeholders have concerns which derive decision topics. In the decision space, *alternatives* are proposed for each decision topic, and based on the ranking of these *alternatives*, *design decisions* are made. As a result of the design decisions, *architectural designs* and the corresponding *design artifacts* are created in the solution space.

Perspective 1.1: Description of problem space.

Software architecture design starts from defining the problem space. Clear and complete statement of problem space is essential to architectural knowledge management. The Core model uses concepts of stakeholder (with its roles and activities), concern (requirements, risks, technical limitations or budget constraints) and decision topics to describe problem space.

Software requirements engineering contributes to the exploration of problem space by finding stakeholders and system requirements. Some decision-centric approaches assume that requirements are given, and other approaches include design concerns other than requirements, such as risks and constraints. We examine the elements of design concern in each approach.

Perspective 1.2: Description of design decisions. Design decisions connect the problem space to the solution space. They are the core elements describing the software architecture.

Our analysis investigates the representation of design decision. Whilst the Core model emphasizes the representation of design decisions in an architecture, the ontology of an architectural design decision model [13] explores the fine-grained characteristics of design decision from different aspects: classification, attributes and relationships (between decisions and upstream artifacts, between design decisions, between design decisions and downstream artifacts). The organization and representation of design decisions therefore have implications on architectural design reasoning and knowledge reuse.

Perspective 1.3: Description of solution space. In the solution space, the design decisions reach their destination—the architectural design. The Core Model abstracts this relationship as “Design Decisions are assumed to influence Architecture Design” and “Artifacts reflects Architectural Design”. Artifact is the notion of architectural document including architectural models, views and documents, etc. We analyze the solution space of each approach by studying the representation of software architecture and the viewpoint/view support provided by each approach.

Perspective 2: Decision making techniques

Decision-making approaches deal with design reasoning through different techniques. They include different steps, work at different architectural levels and support different architectural activities.

Perspective 2.1: Specific steps for decision making.

The Core model uses a *decision loop* for decision making. From this model, we infer that the decision loop consists of the following steps: 1) identify the decision topic from concerns; 2) determine the set of alternatives; 3) determine the criteria that will be used to rank the alternatives; 4) rank the alternatives; 5) choose an alternative which then becomes a design decision; 6) identify new concerns and decision topics that are lead by design decision. Different techniques are applied by the studied approaches to resolve design

reasoning problems in certain steps. We apply these approaches to a common case study and analyze their support for decision making.

Perspective 2.2: Decision making level. Decision making can work at two different levels: (a) at the localized level, decision making deals with a specific decision topic or a decision issue. Each design decision is made by selecting the most appropriate alternative which matches system requirements and the selected alternative becomes part of final architectural design; (b) localized design decisions may be refuted if intertwining design is considered at the cross-cutting design decision level. At this level, revised decisions take into consideration a combination of more than one local decision for eliminating design conflicts or optimizing design. The studied approaches work at different decision making levels.

Perspective 2.3: Support to general design activities. A general model of architecture design is proposed in [16], which classifies design activities as architectural analysis, architectural synthesis and architectural evaluation. The studied approaches provide different levels of support to each activity. Mapping architecting activities with this general model verifies the completeness of each approach.

Perspective 3: Design rationale management

Design rationale is the justification behind design decision. It promotes the communication of stakeholders. Rationale is captured and used in different forms during software engineering [17]. The IEEE 1471-2000 standard [1] considers design rationale as an important element of software architecture. Unfortunately, rationale is often omitted in software architecture documentation [18]. Capturing rationale to avoid knowledge vaporization is the main motivation to represent software architecture as a set of architectural decisions. The studied decision-centric approaches provide different levels of support and different techniques for rationale management, we separate design rationale management into two parts: (a) design rationale capture and (b) design rationale retrieval.

Perspective 3.1: Capturing and documenting design rationale. Software architects would be reluctant to document the design reasoning process and rationale if it is time-consuming and effort-intensive. Aligning rationale capture and documentation with architecting activities is essential to minimize the overheads of rationale management.

Template-based rationale capturing and documenting methodology are proposed in [19]. Some

of the studied approaches improve this methodology by maintaining the relationships between various architectural design entities. Some approaches do not explicitly record design rationale, but document the reasoning processes instead.

Perspective 3.2: Retrieving design rationale. Sharing design rationale supports collaboration in design teams [17], improves software quality [20] and transfers architecture knowledge [21]. The rationale retrieval capability of an architecture design approach facilitates rationale finding, sharing and maintaining during the architecture review and evaluation process.

3. Applying decision-centric methods to an industry case

In order to analyze the procedures and outcomes for each of the nine decision-centric methods, we applied them to a common industry case. The industry case is an ERP system design of a power supply company. Shandong Power Company (SPC) is an asset-intensive company which provides electricity to millions of customers in China. SPC has already an ERP system implemented. To improve the efficiency of its core business operation, SPC decides to extend its ERP system to support the asset maintenance.

The new system has to seamlessly integrate the supply chain management in the existing ERP system and the new asset management module. Design decisions has to be made on how to integrate the purchasing requisition, purchase order and receiving business process, what specific technologies are employed to implement the integration, etc. This case is selected because of its complexity and its cross-cutting and intertwining decisions.

In the case study, we applied each decision-centric method to the same industry case by following the relevant techniques as described in the literature. The application of the techniques to the industry case has yielded different forms of outcomes. We have then used the defined analysis framework to study them.

The results of applying the different methods to the same case study provide interesting reflections on the design decisions that have been made and the steps that need to be taken to make those decisions. We analyze the decision making steps and the results using the three key perspectives discussed in the analysis framework. Due to space limitation, we refer interested readers to the technical report on the details of the case study [22] The analysis results are discussed in the following sections.

4. Applying the analysis framework

In this section, we use the perspectives specified in the analysis framework to assess the applications of different methods to the same industry case.

4.1 Decision-centric architecture knowledge modeling

Perspective 1.1: Description of problem space. All the studied approaches consider design concern as an essential part of architectural knowledge. The common element of concern is the system requirements. Seven of the nine approaches model requirements (functional and nonfunctional) as part of concerns, and the other two approaches (BBNbASM and ArchDesigner) model non-functional requirements (quality attributes) only. The difference is the other elements that constitute architectural concerns. They include environment (AREL), risk (AQUA, ATO, etc), design constraints (AQUA, ADDSS, PAKME, Archium, AREL), scenario (PAKME, Archium), problem (Archium, PAKME) and motivation (Archium, AREL).

Stakeholder initiates *concerns* in the Core model. Five of the nine approaches include the information of stakeholder in architectural knowledge. ArchDesigner evaluates different requirements from different stakeholders on quality attributes. Archium, AREL, PAKME and ADDSS directly relate *stakeholders* with requirements, thus facilitating stakeholder communication, participation and architecture knowledge sharing during the design process.

Perspective 1.2: Description of design decisions. We have found that all approaches document design decisions explicitly in the architectural designs but they organize them in different ways. Seven approaches represent architectural design decisions using a meta model while BBNbASM and ArchDesigner informally define architectural decision as selecting a design alternative. For these seven approaches, when mapping with the three characteristic segments summarized in the ontology [13] of architectural design decision, all of them focus on maintaining architectural decision relationships, and some approaches (i.e. ATO, PAKME, ADDSS, Archium and AREL) cover different types of relationships between architectural elements whilst the others (BBNbASM and ArchDesigner) only manage specific ones. This kind of architectural knowledge allows traceability and dependency analysis of architectural elements. However, the capability provided by each approach varies because of the range of architectural decision relationships covered by them. For instance,

BBNbASM records the impact of design alternatives on quality attributes, but omits the relationship between design decisions and other factors of system concerns. Therefore the tracing from functional requirements to architectural decisions breaks and vice versa. Furthermore, ADDSS and PAKME classify and record the attributes of each decision such as decision status, design history and decision makers, and this additional information provides extra support for understanding and reusing architectural knowledge.

Perspective 1.3: Description of solution space. The manner in which the solution space is described with regards to decision making is important. Archium, ATO, AREL, ASSM and ADDSS capture design decisions and design outcomes in terms of specific views. For instances, Archium and ASSM adopt the component and connector view, ATO uses the IBM Architecture Description Standard, AREL leverages an architecture rationale conceptual model and the UML notation to represent architectural solution. The other four approaches do not prescribe a way of documenting architecture design other than their design decisions.

From the architecture knowledge modeling perspective (i.e. Perspective 1), we found the studied approaches differ in the following ways:

Adaptation to changing requirements: Archium, AREL, AQUA and ATO are adaptable to requirement changes. They allow adjustments to design fragments to support evolving requirements. They also provide traceability from the requirements to architectural designs and vice versa. This capability aids requirement change analysis before implementing the change. ASSM, BBNbASM and ArchDesigner, however, have to pre-define all requirements before applying their reasoning steps, single change in requirements needs the re-analysis of the entire architecture.

Architectural solution description. Five approaches (Archium, ATO, AREL, ASSM and ADDSS) describe the architectural solutions as the results of the design decisions. This knowledge captures both the reasons and the solutions of a design decision. The other four approaches only record design decisions without referencing the associated design solution. Viewpoint specifies the conventions for constructing and using a view [1], only ADDSS and AREL provide viewpoint support in representing architectures.

4.2 Decision making techniques

Perspective 2.1: Specific steps for decision making. Different processes and techniques are used by those approaches to support decision making. BBNbASM

and ArchDesigner employ quantitative method while others use qualitative method.

Archium, AREL and ATO can be applied in an iterative decision making process which corresponds to the “decision loop” in the Core model. They make decisions one by one and use various architectural models or templates to capture and document the decision during the design process.

BBNbASM and ArchDesigner focus on ranking design alternatives. They try to resolve the problem that design decision can cut-across multiple interrelated decision topics. Under this situation, the “divide and conquer” principle of decision making may miss out on decisions that evaluate multiple individual decision topics. Both of them quantify overall architecture value by measuring alternatives’ impact on quality attributes. BBNbASM addresses the inter-dependency relationship between decisions under the situation that one design decision may impact on other decisions. ArchDesigner’s objective function accumulates the value score of each decisions, and take the inter-dependency relationship among design decisions as constraints (conflicting alternatives from different decisions will not be put together). Compared to ArchDesigner, BBNbASM uses Bayesian networks to model design decisions, and is intuitive and easy for reusing. The strong point of ArchDesigner is that the project cost and time constraints are taken into account in the optimization formulation, while BBNbASM only evaluates the predetermined quality attributes. Another weak point of BBNbASM is its lack of prioritizing on quality attributes and design decisions while ArchDesigner uses weighted quality attributes and normalization to measure these two priorities.

ASSM provides an algorithm to automatically generate architecture candidates from the issue solutions and lessen the burden on the architect to explore all alternative combinations. This approach could be helpful for designing a particular part of a complex system where some design decisions are tightly-coupled and cross-cutting. However, in large systems where many more decision issues exist, the algorithm will generate a huge number of candidates, reducing the effectiveness of this approach.

AQUA applies a decision-centric process of finding, evaluating and changing decisions. The decision constraints graph is the most significant technique to support the decision evaluation and decision changing process. However, this approach has not been supported by any computerized tool. How to maintain the decision constraints graph in large software system remains an issue.

PAKME and ADDSS document design decisions with the help of architectural styles and design patterns. As such, they are able to support decision making using general and reusable architectural knowledge.

Perspective 2.2: Decision making level. Archium, ADDSS, PAKME, ATO and AREL work at the *localized level*. Our case study demonstrates that they adopt an iterative process to make one decision after another. ASSM also models localized reasoning as *solution discovering*, but there is no specific techniques suggested for this activity.

BBNbASM, ArchDesigner, ASSM and AQUA work at *cross-cutting design level*. They address inter-related decisions in the case study, including how to integrate and implement different processes in the enterprise supply chain and adopt appropriate technologies to meet the system quality requirements. AQUA and ASSM generate validated architecture in which design decisions meet software requirements and are compatible with each other. BBNbASM and ArchDesigner use quantitative methods to find an optimal architectural design solution.

Perspective 2.3: Support to general design activities. Architectural analysis serves to define the problems the architecture must address, and it generates architecturally significant requirements (ASRs) from concerns [16]. Each approach partially supports this activity. They take concerns as the input of the architectural synthesis activity. However, none of them supports creating ASRs out of general concerns. They all assume that concerns are the same as ASRs, but are they? The architectural synthesis activity proposes architecture solutions based on ASRs. It is the core activities of nearly all studied decision-centric approaches. The only exception is AQUA, which concentrates on evaluating design decisions rather than creating architectural solutions. The architectural evaluation activity validates architecture designs. Six of the studied approaches, Archium, ATO, AREL, PAKME, ADDSS and AQUA, provide support to this activity. Examples are the forward and backward traceability in AREL and Archium, the automatic utility tree generation in PAKME, and the use of decision constraints graph in AQUA in performing architectural evaluation.

From the perspective of design reasoning techniques, the key differences are as follows:

Qualitative and Quantitative Reasoning Techniques: Despite the facts that all approaches deal with decision decisions and organize architectural knowledge, ArchDesigner and BBNbASM focus on quantitative design reasoning techniques, such as the use of AHP (Analytic Hierarchy Process) method in

ArchDesigner. The other seven approaches provide various models or template to facilitate qualitative design reasoning processes, such as the Design Decision Model in AQUA and Archium, architecture rationalization method in AREL and the Artifacts Data Model in PAKME.

Holistic solution versus specific problem solving: the study on perspective 2.2 and 2.3 shows that the studied approaches have different emphasis, i.e. some addresses reasoning issues at the localized level, some try to resolve cross-cutting decision issues, and they supports different activities in software architecting processes. None of the studied approaches address all the issues described in our analysis framework. ASSM is a holistic approach, it addresses both localized level and cross-cutting level design reasoning, it support architectural analysis and synthesis, however, most activities of this approach are still manually performed except for solution synthesizing.

4.3 Design decision rationale management

Perspective 3.1: Capturing and documenting design rationale. Archium uses standard templates to capture design rationale. Design rules, design constraints, consequences and pros and cons are recorded as rationale, indicating why a solution is adopted or discarded. AREL captures architecture design rationale (AR) during the design process, and that is a significant outcome of the software architecture design. ASSM integrates rationale capture with the design process. It documents architecture rationale and issue rationale in terms of pros and cons of solutions. ATO doesn't model rationale in the ontology, however, the wide range of *relationship classes* defined between architectural elements can provide information on why an alternative is selected or not. BBNbASM and ArchDesigner do not explicitly capture design rationale, but the reasoning process, algorithm and the quantitative evaluation provide the design reasoning clues. For instance, the Bayesian network shows an overall influence of design decisions on quality attributes, and it can be reused for the purpose of system maintenance, evolution and design of similar systems in the same domain. In ArchDesigner, the comparative contribution of alternatives to quality attributes and the weighting of quality attributes are kept as rationale.

Although design rationale are captured with different techniques and are in various formats, they facilitate decision making and provide, to various degrees, architectural knowledge reuse and the stakeholder communication.

Perspective 3.2: Retrieving design rationale.

Archium provides a visualization tool to support retrieving architectural decisions and their drivers and rationales. In AREL, ARtrace enables architect tracing the design to the justifications including qualitative rationale, quantitative rationale and alternative architecture rationale, and other root causes, such as requirements, assumptions and constraints (AE). Because all the architecture knowledge is stored in the repository of the Ontology tools, ATO uses a standard query language to retrieve information. BBNbASM and ArchDesigner do not provide retrieval support. AQUA currently has no tool support for its decision model and constraints graph, and any retrieval activity has to be performed manually. PAKME and ADDSS use hyperlink associated with design decisions or design solution to access design rationale.

We have observed that, compared to rationale capture, design rationale retrieval are neglected in some approaches (i.e. BBNbASM, ArchDesigner and AQUA). For large and complex software-intensive systems, although design rationales are captured and documented, their value cannot be realized if there is no proper method to locate the relevant design rationale quickly during the process of architectural evaluation and system maintenance.

5. Findings

Using the three key perspectives to analyze the nine decision-centric design methods, we have summarized the situations of current research in decision-centric design approaches. From our analysis, we have identified a number of gaps for further improvements:

Identifying architecturally significant requirements (ASRs). Architecture design serves as the blueprint of the software system. One principle of software architecting is that software architecture should be defined in terms of elements that are *coarse* enough for human intellectual control and *specific* enough for meaningful reasoning [23]. How to define decision topics is the key to reach this balance of "coarse and specific". This depends on the elicitation, identification and selection of a set of ASRs from general requirements or concerns. The analysis on perspective 1.1 and perspective 2.3 has shown that there is no support on this issue. There are no common accepted criteria on what factors constitute the architectural concerns and current approaches implicitly assume that the requirements that they deal with are automatically ASRs. A clear delineation is important to identify only the relevant architectural requirements. For instance, there is a false positive

scenario - including specific problems that are not architecturally significant; or a false negative scenario - omitting a genuine and relevant architectural requirement.

Making decisions at different levels. According to [2], representing software architectural decisions aims to address the following issues: knowledge vaporizes, cross-cutting and intertwined design decisions are not clearly identified, and design rules and constraints are violated. Through the case study and the analysis of perspectives 2.1 and 3.1, we find that the *knowledge vaporization* problem is addressed by most approaches for they explicitly document design rationale. However, as shown by perspective 2.2, the issue of dealing with design decisions that are cross-cutting and intertwining are not addressed by all the approaches. A comprehensive architectural design approach should support design reasoning at both the *localized level* and the *cross-cutting level*.

Approaches such as AQUA, ArchDesigner and BBNbASM perform reasoning at the cross-cutting level, and approaches such as Archium and AREL perform reasoning at the localized level. These two levels of decision making are complementary in the architecting process. For example, AREL is a rationale-based architectural design approach, and its Architecture Rationale (AR) elements can accommodate the BBNbASM or ArchDesigner approach as the quantitative rationale. This flexibility allows AREL to accommodate, say the ArchDesigner technique, cross-cutting design decision making. Another example is ATO, which can be adopted by ASSM to document the relationships between design alternatives, and implement the solution synthesizing algorithm based on the ontology platform.

Applying decision making steps. Not all decision-centric approaches have suggested how to carry out decision making steps. including Archium, AREL, PAKME and ADDSS. We have referred the *decision loop* in the Core model to apply these approaches. We found that there are missing details in the decision making process. For instance, *identifying decision topic* is a key step in the Core model. However, there are no techniques, criteria or steps in these approaches to direct a designer to do this. A similar issue exists in *identifying design alternatives*. Such incomplete support on *decision loop* may lead to the neglect of critical design concerns and potential design alternatives.

Reusing design rationale. From perspective 1, we have observed that some of the studied approaches do not model the architecture knowledge completely, no matter in the problem space, decision space or solution

space. This makes it difficult to relate design rationale to the other relevant architectural information. For example, the lack of functional requirements in ArchDesigner makes it difficult for architectural stakeholders to explain what have influenced specific decisions. The absence of architectural representation in AQUA will make it impossible for the software maintainer to analyze the impact of architectural changes in components or connectors using design rationales.

Supporting software life-cycle with architectural knowledge. The Core model encompasses the fundamental concepts of architectural knowledge. From perspectives 2.3 and 3.1, we observe that architectural knowledge (i.e. design rationale) is captured to support architectural design. Other software life cycle activities, like requirement engineering, detailed design, testing, implementation, deployment and maintenance, can also benefit from architectural knowledge. There is support for some of these activities provided by certain approaches. For example, PAKME uses the concept *project* to support system implementation, and ATO uses the *implication class* to relate architectural decision with the design of system deployment. However, the support to the software life-cycle, in general by these approaches, is limited. Some of these approaches do not specifically describe how the captured architectural knowledge is retrieved and then reused. There is potential for decision-centric approaches to extend their architectural knowledge management to support more activities in the software life-cycle, therefore increasing the value of documenting the design decisions and rationale in software architecture.

6. Conclusion

In this paper, we have analyzed nine decision-centric architectural design approaches to investigate their support for design reasoning from three perspectives: (1) architectural knowledge modelling; (2) decision making techniques; (3) design rationale management.

We have applied each of the decision-centric approaches to the same industry case to assist with the analysis [22]. Through the applications of these approaches, we have observed that although they provide techniques to perform or represent design reasoning, they are based on different premises and aspects of design reasoning. We have found that most approaches assume that ASRs are given, which do not need justifications, and certain approaches consider architectural decisions are made at one level. By

challenging these assumptions, we suggest that there is a need to investigate deeper into the fundamentals of what design reasoning should be based on. We have also identified a number of areas that calls for further improvement, including identifying architecturally significant requirements, improving current design reasoning processes and extending architectural knowledge management to support the software life-cycle.

7. References

- [1] IEEE, "IEEE Recommended Practice for Architectural Description of Software-Intensive Systems (IEEE Std 1471-2000)," 2000.
- [2] J. Bosch, "Software architecture: The next step," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. vol. 3047, 2004, pp. 194-199.
- [3] A. Jansen and J. Bosch, "Software Architecture as a Set of Architectural Design Decisions," in *Software Architecture, 2005. WICSA 2005. 5th Working IEEE/IFIP Conference on*, 2005, pp. 109-120.
- [4] A. Jansen, J. van der Ven, P. Avgeriou, and D. K. Hammer, "Tool Support for Architectural Decisions," in *Software Architecture, 2007. WICSA '07. The Working IEEE/IFIP Conference on*, 2007, pp. 4-4.
- [5] A. Akerman and J. Tyree, "Using ontology to support development of software architectures," *IBM Systems Journal*, vol. 45, pp. 813-825, 2006.
- [6] A. Tang, Y. Jin, and J. Han, "A rationale-based architecture model for design traceability and reasoning," *Journal of Systems and Software*, vol. 80, pp. 918-934, 2007.
- [7] T. Al-Naeem, I. Gorton, M. A. 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*, 2005, pp. 244-253.
- [8] H. Zhang and S. Jarzabek, "A Bayesian Network approach to rational architectural design," *International Journal of Software Engineering and Knowledge Engineering*, vol. 15, pp. 695-717, 2005.
- [9] H. Choi, Y. Choi, and K. Yeom, "An Integrated Approach to Quality Achievement with Architectural Design Decisions," *Journal of Software*, vol. Volume 1, pp. 40-49, 2006.
- [10] X. Cui, Y. Sun, and H. Mei, "Towards Automated Solution Synthesis and Rationale Capture in Decision-Centric Architecture Design," *Seventh Working IEEE/IFIP Conference on Software Architecture*, pp. 221-230, 2008.
- [11] M. A. Babar and I. Gorton, "A tool for managing software architecture knowledge," in *Proceedings - ICSE 2007 Workshops: Second Workshop on SHaring and Reusing architectural Knowledge Architecture, Rationale, and Design Intent, SHARK-ADI'07*, Minneapolis, MN, 2007.
- [12] R. Capilla, F. Nava, S. Pérez, and J. C. Dueñas, "A web-based tool for managing architectural design decisions," *Ist ACM Workshop on SHaring ARchitectural Knowledge (SHARK)*, 2006.
- [13] P. Kruchten, "An ontology of architectural design decisions in software-intensive systems," *Intern. Workshop on Software Variability Management*, 2004.
- [14] J. Tyree and A. Akerman, "Architecture decisions: Demystifying architecture," *IEEE Software*, vol. 22, pp. 19-27, 2005.
- [15] R. C. De Boer, R. Farenhorst, P. Lago, H. Van Vliet, V. Clerc, and A. Jansen, "Architectural knowledge: Getting to the core," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. vol. 4880 LNCS Medford, MA, 2007, pp. 197-214.
- [16] 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, pp. 106-126, 2007.
- [17] A. H. Dutoit and B. Paech, "Rationale Management in Software Engineering: Concepts and Techniques," pp. 1-48, 2006.
- [18] A. Tang, M. A. Babar, I. Gorton, and J. Han, "A survey of architecture design rationale," *Journal of Systems and Software*, vol. 79, pp. 1792-1804, 2006.
- [19] P. Clements, D. Garlan, R. Little, R. Nord, and J. Stafford, "Documenting software architectures: Views and beyond," in *Proceedings - International Conference on Software Engineering*, Portland, OR, 2003, pp. 740-741.
- [20] A. Tang, M. H. Tran, J. Han, and H. v. Vliet, "Design Reasoning Improves Software Design Quality," *QoSA 2008, LNCS 5281*, pp. 28-42, 2008.
- [21] P. Avgeriou, P. Kruchten, P. Lago, P. Grisham, and D. Perry, "Sharing and reusing architectural knowledge - Architecture, rationale, and design intent," in *Proceedings - International Conference on Software Engineering*, Minneapolis, MN, 2007, pp. 109-110.
- [22] W. Bu, A. Tang, and J. Han, "An Analysis of Decision-centric architectural design approaches," Swinburne University of Technology http://www.swinburne.edu.au/ict/research/documents/SUTIC_T_TR2009_01.pdf, 2009.
- [23] R. Kazman, L. Bass, and M. Klein, "The essential components of software architecture design and analysis," *Journal of Systems and Software*, vol. 79, pp. 1207-1216, 2006.