Faculty of Information & Communication Technologies

An analysis of decision-centric architectural design approaches

Technical Report: SUTICT-TR2009.01

Wanfeng Bu, Swinburne University of Technology
Antony Tang, Swinburne University of Technology
Jun Han, Swinburne University of Technology
19 Jan 2009

Table of Contents

Table of Figures

Faculty of Information & Communication Technologies

An Analysis of Decision-centric Architectural Design Approaches

1. Introduction

Software architecture provides a blueprint for a software-intensive system. 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 design only depicts components and connectors, but omits the design decisions and design rationale. This may lead to 1) costly support efforts 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 focus on providing decision support and documenting 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 organisation and documentation of design decisions explicitly; 3) integrate techniques/methodologies in the design reasoning process; 4) capture and document design rationale. Some of these approaches include Archium [3, 4], Akerman's Ontology [5], AREL [6], ArchDesigner [7], Bayesian Network based Alternatives Selection Method [8], AQUA [9], and Automated Solution Synthesis Method [10]. We consider them to be decision-centric architecture design approach. In addition, we also study PAKME[11] and ADDSS[12] for they provide tool support for knowledge management and decision making by maintaining an architecture knowledge repository (i.e. architectural styles and design patterns) even though they do not provide specific design techniques.

All these software architecture decision-centric approaches share some common characteristics. They either capture design rationale and/or provide methods to support design reasoning, they use requirements as a basis to support reasoning and they employ different techniques to justify a design. However, they also differ in many ways. The design reasoning process they employ differs greatly, and as a result the reasoning process and the resulting design rationale varies. Some approaches address high-level architectural design whilst others also encompass the design at the programming level. The main objectives of this study are two folds. Firstly, to provide a bird's eyes view of this developing field, and to differentiate the differences and the similarities of these approaches. Secondly, through the analysis, we aim to identify the strengths and weaknesses of theses approaches to uncover new research alignments and questions.

In this report, we use three key perspectives to analyse 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. In section 2, we explain the framework that is used to analyse these approaches. Section 3 briefly introduces each approach. A common case is used to illustrate the features of each approach in section 4. Section 5 uses the framework to analyse and compare these approaches. Section 6 presents our findings and Section 7 concludes this report.

2. Analysis framework

During the study, we have found that different approaches have made different assertions on what design reasoning is fundamentally about. Although there is also much similarity between them, we recognise the need to have a common framework to support the analysis. A viable common framework is the IEEE 1471-2000 standard on software architecture description [1]. However the reference to design reasoning is limited in the standard. A more recent definition is provided by the Core model [13], it describes the key concepts of architectural knowledge and design reasoning.

We base our analysis and comparison on three key software architecture perspectives: software architecture knowledge, decision making techniques and design rationale. 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* deals with the capture and retrieval of design rationale to support software architectural design.

Perspective 1: Decision-centric architectural knowledge modelling

Capturing and representing architectural knowledge is an essential component of software engineering. The IEEE 1471-2000 standard [1] suggests a conceptual framework for architectural description, however the references to design decisions and design reasoning are limited in this standard. A more recent definition is provided by an architectural knowledge model, i.e. the Core model [13] . It describes the key elements of architectural knowledge and design reasoning with minimalistic but complete set of concepts [13] .
The elements in the Core model (Figure 1) 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 that derive decision topics. For each decision topic, alternatives are proposed, and in turn design decisions are made based on the ranking of these alternatives. Design decision in the *decision space* has influence on architectural designs and may lead to new concerns where a new decision loop begins. As a result of the design decisions, architectural designs and the corresponding design artefacts are created in the *solution space*.

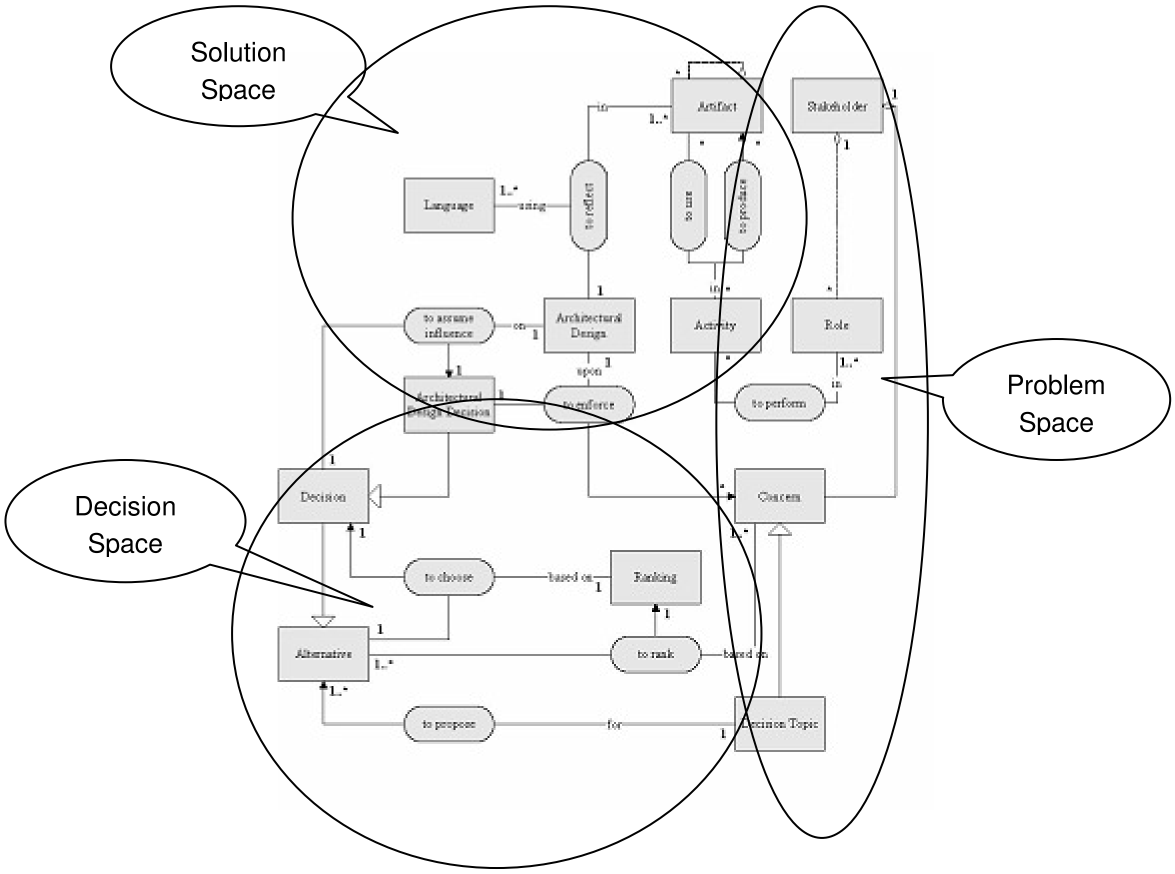


Figure 1 Core Model

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) and Concern, to abstract problem space. The two concepts are compatible with the definitions in the IEEE1471-2000 standard [1]. Stakeholders are "individual, team, or organization (or classes thereof) with interests in, or concerns relative to, a system". Concerns are "those interests which pertain to the system's development, its operation or any other aspects that are critical or otherwise important to one or more stakeholders". It is a broad concept which could be used to abstract requirements, risks, technical limitations or budget constraints [13].

Software requirements engineering contributes to the exploration of problem space. There are several challenges in this domain. One is that not all requirements will be relevant to the architecture, the architect should refine architectural concerns into architecturally significant requirements (ASR) that are specific in terms of desired system properties [14]. Secondly, requirements impact each other, sometimes they even conflict with each other. Thirdly, requirements are subject to change and managing change continues to be a fundamental problem[15].
In the analysis, we investigate how stakeholders' concerns are modelled and organised to overcome the above challenges, how each approach supports the change of requirements.

Perspective 1.2: Description of design decisions

Design decisions connect the problem space to the solution space. They are the core elements describing software architectures. The Core Model uses a “decision loop” to illustrate this space.

First, our analysis will investigate the representation of design decision. Whilst the Core model emphasizes the representation of design decisions in architecture, the ontology of an architectural design decision model in [16] explores the characteristics of design decision from the aspects of classification, attributes and various relationships(between decisions and upstream artefacts, between design decisions, between design decisions and downstream artefacts). We will study how each approach classifies design decisions, what attributes they assign to design decision and what kinds of relationships they document and maintain.

Second, we will try to find what specific architecting problems are resolved by explicitly documenting design decisions. According to [2], representing software architectural decisions aims to address the following issues:

- Knowledge vaporizes (KV): the knowledge about the “what and how” of the software architecture is quickly lost [2] due to the fact that design reasoning process is not recorded;
- Cross-cutting and intertwined design decisions are not clearly identified: architecture design decisions typically are cross-cutting (affect multiple components and connectors at the same time), and often become intertwined with other design decisions (CC-INTER) [2]; changing or removing existing design decisions requires representing the complex relationships among design decisions clearly because the impact analysis of an architectural adjustment relies on the understanding of the influential relationships .
- Design rules and constraints are violated (RULE): During the evolution of software systems, designers, and even architects, may easily violate the design rules and constraints imposed on the architecture by earlier design decisions[2].

We check whether the above three problems (KV, CC-INTER, RULE) are addressed/resolved in each approach by representing design decision as a “first-class entity”.

Perspective 1.3: Description of Solution Space

In the solution space, architectural designs implement all design decisions. The Core Model abstracts this relationship as “Design Decisions are assumed to influence Architecture Design” and “Artefacts reflects Architectural Design” (see Figure 1). Artefact captures the notion of architectural document including Architectural Model and View [1], document, source code, etc. The purpose of documenting software architecture is for 1) early analysis, 2)defining work assignments, and 3)guiding system evolution[17]. An architecture design approach should provide facilities and convenience to achieve these goals.

We will investigate the solution space of each decision-centric design approach by studying how software architecture is described in it. Such analysis would require the understanding of the architectural description from a number of perspectives:

- The viewpoint/view support provided by each approach
- Tight integration between decision decisions and architectural designs and their support on forward and backward tracing.

Perspective 2: Decision Making Techniques

Design decision are always hard to make because they are cross-cutting and intertwining [2]. It is quite often that design decisions have great impact on non-function requirements, or disparate design decisions impact on the same architecture components [2]. Decision-making approaches try to address how design decisions are made and how reasonable they are through different techniques.

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. However, different approaches have their own emphasis on these steps. Some approaches apply specific techniques to resolve certain design reasoning problem. For example, Bayesian belief network is used in one approach to evaluate the influence of design decisions on quality attributes because the impact is usually difficult to determine and quantify. We will use a common case to illustrate the principle of each approach and analyse their supports on design reasoning in section 5.

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 all selected alternatives becomes part of final architectural design; (b) localised 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 and optimizing design.

Perspective 2.3: Supports to a general design activities

A general model of architecture design is proposed in [18], which classifies design activities as architectural analysis, architectural synthesis and architectural evaluation. The architecture design approaches provide different levels of support to each activity. Mapping architecting with this general model verifies the completeness of each approach. Following considerations are included in our analysis and comparison: How does each approach select architectural significant requirements based on general concerns (architectural analysis)? How are candidate architectural solutions found (architectural synthesis)? Are there any tools or steps to support architecture evaluation? How are software evolution supported by the techniques?

Perspective 3: Design decision rationale.

Rationale is the justification behind decisions. It is captured and used in many different forms during software engineering [19]. Design rationale, together with design rules and design constraints, should be explicitly documented as part of the design, so they can be understood by all the stakeholders. The IEEE 1471-2000 standard [1] proposes that design rationale be an important element of software architecture. Unfortunately, rationale are often omitted in software architecture documentation [20]. Capturing rationale to avoid knowledge vaporization is the main motivation to represent software architecture as a set of architecture decisions. The studied decision-centric approaches provide different level of support for rationale management. We analyse them from two perspectives: (a) design rationale capture and (b) design rationale retrieval.

Perspective 3.1: Capturing and documenting design rationale

Software architect would be reluctant to document design reasoning process and rationale if it is time-consuming and effort-intensive. Aligning rationale capture and documentation with the design activities is essential to decrease the overhead of rationale management.

Template-based rationale capturing and documenting methodology are proposed in [21]. Some of the studied approaches improve this methodology by maintaining the relationships between various architectural design entities. Some of the approaches do not explicitly record design rationale, but the reasoning processes are documented.

Perspective 3.2: Retrieving design rationale

Sharing design rationales supports collaboration in design teams [19], improves software quality [22], transfers architecture knowledge [23]. Rationale retrieving capability of architecture design approach facilitates rationale identification and sharing during the architecture review and evaluation process. We will study what kind of support each approach provides to retrieve design rationales.

3. Overview of decision-centric architecture design approaches

3.1 Archium

Archium (1) regards architectural decisions as the first class entity in software architecture and (2) defines the relationship between these design decisions and software architecture. It is based on a meta-model consisting of three sub-models: an architectural model, a design decision model and a composition model [3]. As first-class entities, design decisions are represented using design decision model, in which decision context are defined. Software architecture are described using the architecture model, which is similar to Component and Connector view [21]. The Composition Model is responsible for relating the decision decisions with relevant elements in the Architectural Model. To describe the architectural element, express architectural decisions and define composition rules, Archium introduces a specific ADL based language which can be compiled by the Archium tool into Java byte code then run in the Archium run-time platform. Design fragments can be composited to the architecture which is visualized in this platform. Architects can inspect the consequences of an architectural decision with the help of this visual environment [4].

3.2 Akerman and Tyree 's Ontology

This approach focuses on architectural decisions and represents architecture through ontology. Ontology in this approach has four major segments: architecture assets, architecture decisions, stakeholder concerns, and an architecture roadmap[5].

The *architecture assets segment* of the Ontology adopts a simplified version of the IBM Architecture Description Standard. It includes *subsystems, components, interfaces, data and node*. The *concern segment* captures the stakeholder's concerns, including *business need* (business goals, objectives, or issues), *risk, change case* (future requirements), *quality* (non-functional requirements such as performance, reliability, etc.), and *capability* (functional features of the system). The *Decision Segment* is the key concept in this approach and it connects other three segments. In this segment, *alternatives* can be developed based on information in the *concern segment*, and in turn an *architecture decision* can be made by selecting an alternative. *Implication* keeps the result analysis of selecting a particular alternative, it generates a new *concern*, therefore a new decision iteration is needed to address this new concern (Implication). The source of Implication may also include the requirements incurred by other decision or some external systems. The *roadmap segment* aims at implementing the software architecture design, it provides direction on migrating to the target system. It addresses the issue that large or complex architecture decision may be implemented by multiple projects.

To build the instance of this ontology for a particular system, this approach suggest a five-steps methodology which is similar to TOGAF(The Open Group Architecture Framework), including 1) capturing stakeholder concerns, 2) analysing the current architecture, 3) defining the target architecture, 4) conducting a gap analysis and producing the roadmap, and 5) validating the architecture.. It also provides a structured repository for on-demand view creation which tries to bridge the ontology with the traditional view-based approach.

This approach is developed on an open source ontology tool Protégé which provides repository support. The architecture information is compatible and can be accessed by other model tools. Also, impact analysis can be performed using SQL query on the database schema.

3.3 AREL

AREL is a rationale-based architecture design approach focus on the design decision reasoning process. In this approach, decision rationales are regarded as part of architectural knowledge and a part of architecture design. The core of AREL is its architecture rationale conceptual model(ARM) which depicts the relationship between motivational reason, design rationale and design object [6]. It uses UML notation to represent the architecture. In AREL ARM model, there are three kinds of key elements: AE, AR and ARtrace. AE is an architecture element which is the input (motivation AE) or the outcome(design outcome AE) of a decision. Existing design elements created in UML can be modelled as AEs. AR encapsulates the architecture design rationale and has its input AEs and output AEs. From this perspective, AR can also be considered as the decision point, and the output AE is the design decision outcome. ARtrace is a stereotyped association representing the relationship between AEs and ARs.

In AREL, decision rationale and dependency are explicitly documented. It also adopts viewpoints to group architecture elements to models and views, which provide the compatibility with IEEE 1471-

2000 standard. Using AREL and an eAREL links, which take advantage of the UML's dependency relationship, traceability and evolution support are provided.

3.4 AQUA

AQUA is an integrated design-decision based architecture design approach. In its decision model, there are three characteristics: (1) decisions are made to satisfy requirements; (2)architecture is described as a set of decisions; (3) Decision Constraint Graph provides dependency relationships among design decisions [9].

AQUA's decision-centric process involves architectural evaluation and transformation. Architectural evaluation is to find and evaluate decisions, which is decision-centric with respect to system requirements. Architecture transformation will apply changes in order to reduce potential defects incurred by competing decisions or decision can not satisfy certain requirements. This change will be subject to change impact analysis using a Decision Constraint Graph before the change can real be applied.

AQUA also proposed detailed steps to perform software architecture evaluation and transformation. However, these steps are not supported by any tools.

3.5 ArchDesigner

ArchDesigner is a quantitative quality-driven architecture design approach that tries to find the best architecture which satisfies conflicting stakeholders' quality goals, competing architectural concerns, and project constraints such as time and cost limitations [7]. The approach uses optimization techniques to recommend a combination of local alternatives in a set of decision topics.

ArchDesigner has three steps. First, all design decisions and associated alternatives for each decision are identified. For each decision topic, the alternatives' impacts on each quality attributes are quantified using MADM (Multiple Attribute Decision Method, a relative weighting method). Quality attributes are weighted based on opinions of all stakeholders using the same method. The impact of each alternative on the whole architecture in each decision topic, quantified by a value score, can be computed by multiplying the two values: quantified alternative quality support and weighted value of quality attributes. Second, the computed values are normalized so that they can be added together reasonably because different decision make different contribution to architecture. Thirdly, applying optimization equation with the consideration of other system constraints to find a globally optimal solution, which is a set of selected alternatives.

3.6 BBN-based Design Alternative Selection

This approach uses BBN(Bayesian Belief Network) to support design decision-making. It is based on the assumption that the impact of design decisions on quality attributes is undeterministic, the proper way to describe this impact is to assign a probability value between 0 and 1, rather than a deterministic value which is 0 or 1[8]. A Bayesian network is used to quantify the complex relationships among design decisions and quality attributes to reach optimized software architecture.

The two sets of parameter in this approach are the design decisions and the quality attributes. A Directed Acyclic Graph (DAG), consisting of design decisions and all the quality attributes, defines the whole design space. In DAG, design decisions and quality attributes are all Nodes, and the influences of design decisions on quality attributes and the influences between design decisions are

Edges. Meanwhile, the influence is measured by a probability which is given by domain expert, each node is associated with a Conditional Probability Table (CPT) to store the conditional probability.

Once the Bayesian network is built up, a Bayesian network toolkit can be used to calculate the posterior probability value of each quality attributes, then different designs (decisions have different values) can be compared with the probability of quality values, the higher the probability, the more possible this quality attribute may be achieved.

3.7 Automated Solution Synthesis

This decision-centric architecture design approach aims at bridging the gap between software requirements and architectures with design decisions[10]. Based on a decision-centric meta-model, this approach will capture the rationale while solutions are being explored and decisions are being made.

It follows a 4-step process. The first step is the issue eliciting process which elicits the issues from the software requirements (functional and non-functional). An issue is an architecturally significant requirement which addresses one specific aspect of requirements or any other high level considerations. The second step is the Solution Exploiting process which finds candidate solutions for each issue and the advantages and disadvantages of each solution are kept (as pros and cons). The third step is solution synthesizing, which automatically synthesizes the candidate architecture solutions from various issue solutions (find all combinations that can meet the requirements of all issues). The fourth step is architecture deciding, in this step, the architect evaluates each candidate and select the target architecture solution as the output.

The core element of this approach is the solution synthesizing algorithm, which automatically generate a set of candidate architecture decisions based on all possible combinations of the issue decisions. Generalized, conflictive and independent relationship between issue solutions are used to minimize the number of candidate architecture solutions.

3.8 PAKME

PAKME is web-based architecture knowledge management tool that is aimed at providing knowledge management support for the software architecture process [11].

PAKME provides four main services: knowledge acquisition, knowledge maintenance, knowledge retrieval and knowledge presentation. Knowledge acquisition service provides various forms and editing tools to enter new generic or project specific knowledge into the repository. The architecture knowledge could be a scenario, design option, design rationale, which are project specific, also it could be a design pattern, case study, tactics achieving quality, which are generic architectural knowledge. Knowledge maintenance service provides function to update knowledge repository. Knowledge retrieval service helps architecture users to find relevant information. Keyword search, string combination search and navigational search are the three main retrieving mechanisms. Knowledge presentation service presents architecture knowledge with various templates and representation mechanisms (i.e. quality attribute utility tree).

PAKEME defines an architecture knowledge retrieve/reuse/revise/retain cycle, with the above four services, PAKME can facilitate this use/reuse cycle.

PAKME is built on top of Hipergate (an open source groupware) and modifies its data model, presentation and business logic tiers for adding features required to manage architectural knowledge. PAKME supports multi-user and multi-project collaboration. Taking advantage of PAKME'S web architecture, geographically distributed stakeholders can share architecture knowledge.

3.9 ADDSS

ADDSS is a web-based design decision management tools. ADDSS is based on a meta-model of architecture design decisions to implement “decision view”[24]. This decision view extends the “4+1” model [25] and acts as an intermediate step between requirements and the “4+1” views. Using the meta-model, ADDSS represents the relations between the decisions, the architecture, and stakeholders involved in different architectural views and the iterations performed in the design process [26]. The main concepts in this model are: stakeholders, decisions, requirements and architectures. What the metal model emphasizes (also the decision view) is the iterative process by which design decisions are formalized. The intermediate architecture designs are kept (as the architecture design history) within each iteration.

The ADDSS tool allows the storage of different projects and architectures. Each project may have one or several architectures. Registered users may have different permissions for accessing the information stored in the tool.

The ADDSS tool (ver 1.0) is developed using dynamic HTML and PHP with the Apache 2 web server. The information about design decisions is stored in a MySQL database. The system uses the Gd2 graphics library for producing thumbnail images of the architectures uploaded by the users.

4. Applying the decision-centric approaches: A case study

In this section, we use an industry case as an example (section 4.1) and apply to it the decision-centric architecture design approaches to gain some insights into these approaches. The industry case is an ERP system integration design for a power supply company. The decision-centric methods are applied by following the steps that are necessary to that method. It illustrates the decision making steps used. Therefore, it represents the decision making perspectives described in section 2. Other perspectives such as architectural knowledge modelling and the design rationale are also discussed in this section.

4.1 Case Introduction

Shandong Power Company (SPC) is an asset-intensive utility company which provides electricity to millions of customers in China. It has over 20 district business units, with each being in charge of one geographic area. SPC's headquarter control on district business units is at the planning and budgeting level, and each district business has a high degree autonomy to run business as long as it can meet the budget objective set by the headquarter. The top management of SPC is aware that there is a lot of room to cut cost in the district business units and it needs to enhance the control of the asset management which is the core business of each business unit and where expenses can be saved. SPC decides to take advantage of information technology to achieve the savings.

SPC has already an ERP system implemented. The system functionality covers financial management, supply chain management and human resources management. The financial management sub-system consists of General Ledger (GL), Account Payable/Receivable (AP/AR), Project costing (PC) and Budget control (CC), etc. The supply chain management sub-system covers the business process of Purchase Requisition (PR), Purchase Order (PO) and Receiving (RCV). The human resources sub-system includes the human resource profile module (HR) and the payroll module. All these modules are integrated in an integrated PeopleSoft system (PS). However, to manage the assets across the whole company and to reduce equipment down time and the equipment maintenance cost, there is still other software modules needed. SPC plans to extend its ERP system to the asset management domain. Unfortunately, the ERP package SPC is using (PS) doesn't possess the functionality that meet this requirement.

The functional requirements of this asset management system include: 1) profile management for each equipment(where there are millions), 2) work management covering the activities of emergency maintenance, routine maintenance and preventive maintenance, 3) cost tracking on how maintenance expenses are distributed to work orders or equipments, 4) warehouse management. The sources of the cost are material used in the maintenance, the labour cost and outsourcing contract fees. The non-functional requirements include system maintainability, performance, easy to use, quick implementation (easy to implement), low project cost and security. System stakeholders are site asset managers, financial analysts, purchasing managers, inventory managers and IT system managers, etc.

From the outset, there are two possible solutions. One is to leverage PS's re-development capability and build relevant components to extent the current ERP system. The advantage is that the work management module can be seamlessly integrated with the supply chain module of PS. However, each business unit has different business processes and rules to manage their businesses, which makes the development project much more complex. On the other hand, system maintainability may became a problem because future update of the PS system may have uncertain impact on this customization part of the system. There could be much verification needed before applying system patches.

Another alternative is to use a commercial EAM (Enterprise Asset Management) system. According to the investigation on several EAM packages by the IT department, this kind of system will meet more than 95% percent of SPC's business requirements, and the system implementation are proved to be faster than developing an in-house system. An EAM solution called MAX is a preferred candidate.

At the beginning of this project, most stakeholders prefer the second alternatives, but some design difficulties had been predicted. One is about selecting centralized deployment or decentralized deployment architecture for the MAX system, another is on how the two systems are to be integrated. If MAX is to be implemented, it seems that the supply chain process will inevitably across the two systems. Currently this process is implemented in PS but users complain that the system is quite slow. The reason is that the user number of PR is quite big and they are distributed among all the district business units. This incurs a huge amount of concurrent query on one big centralized ITEM table over the corporate WAN network. According to SPC's business process, this supply chain is divided into 3 parts, purchasing requisition (PR), purchasing order(PO), and Receiving(RCV). The process owners of PR are the equipment maintenance managers who decide when to buy and what to buy. PO are supervised by the purchasing department which decides from whom to buy(sourcing), RCV are performed by inventory managers. Together with QA professionals, they decide wether or not the material meets the quality requirements defined in the PO or purchasing contracts.

One issue in planning the interface is the inventory management. In the PS financial sub-system, there is no inventory defined and the material costs are charged to the project defined by the finance department. That means that the materials are considered to have been consumed by a specific project once received. This is required by the “zero-inventory” strategy adopted by SPC, although in reality materials are stored in the inventory warehouses located in each business unit. Both of the two systems (PS and MAX) have the functionality of inventory management.

If the inventory module is in MAX, the RCV in PS needs to be modified to be capable of updating the Inventory in MAX. An alternative is migrating RCV activities to MAX, and synchronize back to PS for the purpose of generating accounting entry information. In addition, the PR and PO parts also have several interfacing alternatives.

The cross-cutting design issues above prevent the project team to decide on using MAX. Extending the current ERP system or using the mature MAX system is a general architectural decision which has to be made at the beginning of system design, and it will result in different kinds of system architecture. The advantages and uncertainties illustrated above are based on the initial analysis by the IT department. In the case study, we focus on the key architectural decisions for selecting an appropriate technology. We omit other decisions topics of MAX function requirements because they are relatively isolated decisions that do not add to this illustration.

4.2 Archium

Archium’s architecture model can be used to present the current PS system architecture, then problems are analysed within its design decision model followed by making a design decision by exploring all alternatives, finally the design fragment of this decision is composed to the existing architecture with the help of Archium composition model.

Archium doesn’t explicitly specify a decision making process, we refer the *decision loop* in the Core model to apply this approach to the case study. Similar issues exist in applying AREL, PAKME and ADDSS.

Step 1-- Present original design with Archium Architectural Model
Archium’s Architecture Model adopts the concepts of the Component & Connector view[3]. The key concepts include Component Entity, Delta, Interface, Port and Connector. The top part of Figure 2 shows the original PS system using its Architectural Model. PS system is a three-tier application consisting the Web, Application/Transaction, and Database tiers. Inside the Application/Transaction tier, there are several business modules that will be relevant to this case (for simplicity, other modules are omitted) and are represented by “component”. These components are linked by “connectors” to represent the business relationships between them. For instance, when material receiving(RCV) is committed, account payable information will be generated via a “connector”, which is actually a PS application engine program (a mechanism that transfers data between modules in PS).

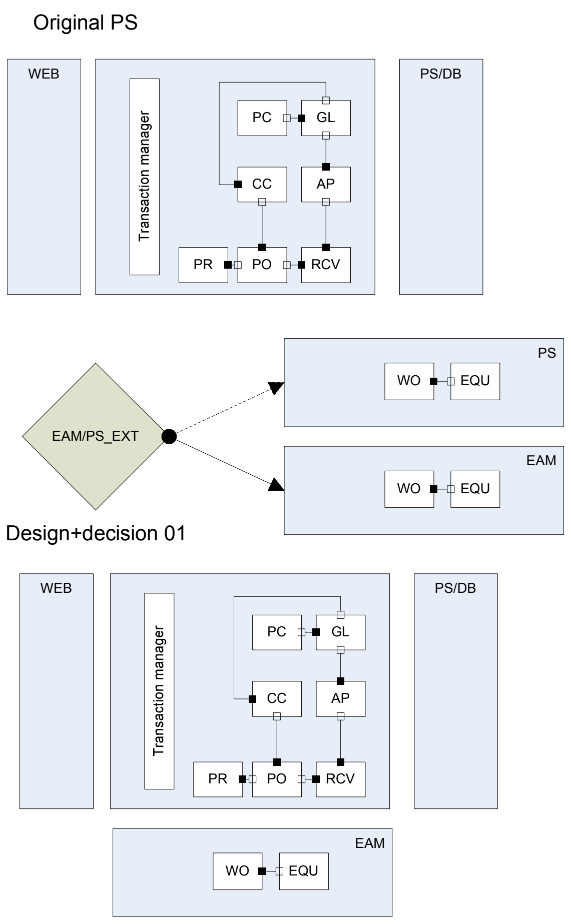


Figure 2 Archium - Decision and Design

Step 2-- Present Architecture decision and its design fragment with Design Decision Model
In Archium, Design decision is a first class concept. The core element of decision is the Problem element, together with the Motivation and Cause element, describes the problem this decision aims to resolve. Decision also defines the solutions considered. For each solution, there are several elements documented: Description, Design rule, Design constraints, Consequences, Pros and Cons. Table 1 shows the template of Archium design decision with our case. Using this template, the design decision, using MAX or extending current PS system, is defined and the two alternatives are evaluated based on their pros and cons.

Decision: MAX or PS_ext	
Problem: Current PS system does not cover the equipment management, which is an essential part of corporate business. Cost management in the maintenance business is at a coarse level and the headquarter lacks control on the procedure and cost.	
Motivation: improve the asset management efficiency in the maintenance processes.	
Cause: The current supply chain system charges the cost of purchasing on corporate level projects. The purchase requisitions delivered by district business units are based on those projects which are too broad to trace how the cost are distributed among the equipment maintenance activities.	
Potential Solutions:	
Alternative: MAX	Description: Use the commercial MAX system to satisfy management requirements. Work orders are implemented in the district business unit, and materials and labour costs are recorded in each work order, providing cost traceability. MAX will run independently but

	synchronize required data to and from PS.
	Design Rules Two system shares master data, such as items, suppliers, and projects
	Design Constraints An intermediate table will be used to transfer master data and transaction data to ensure system stability.
	Consequences PS and MAX will be seamlessly integrated. Inventory management has to be implemented to support cost allocation.
	Pros Mature solution in asset maintenance area; Quick implementation, best practises are available to improve business process, etc. Flexible implementation strategy.
	Cons The volume of data transferred between the two systems will be huge and the integration task is heavy; some users will have to learn another system.
Alternative:PS_EXT	Description Extend the functionality of PS, New component will be developed, including Work Order Management and Equipments, current work flow in the supply chain management will be adjusted to meet the new requirements.
	Design Rules Any modification to PS will through the programming tools provided by PS.
	Design Constraints Related data tables are created in PS database and have to follow the PS metadata policy.
	Consequences New components are developed in PS.
	Pros highly integrated
	Cons Long development phase; the performance quality are difficult to pre-estimated; Long-term business requirements are difficult to meet with in-house application. The current PS platforms (server, db and storage) will need to be re-sized to accommodate the new module. Bring difficulty to future PS upgrades.

Table 1: use Archium's architectural design decision model to define a decision

A design decision will result in a design fragment, which defines the relationship between the design decision and architectural concepts. As in this case, the D01 decision have two solutions which result in two different design fragments as show in the middle part of Figure 2.

Step 3 -- Using Composition Model to relate the changes of the design decision model to the elements of the architectural model
In this case, design alternative MAX is chosen, the lower part of Figure 2 illustrates that the design fragment of this decision is composed into the architecture. Archium uses the Composition Model to compose design fragments automatically. This model consists of Composition Technique, Composition Configuration and Design Fragment Composition. In Figure 3, for example, design fragment 1 illustrates the design of using component interface to "access PS project information in MAX" (a decision). This new function can be defined as a "delta" of MAX. The Composition Techniques describe the way in which "access Project definition"(delta of MAX) changes a port of a

component entity(MAX). On the PS side, an API (a PS interfacing package) has to be installed on the PS Web Server to provide inbound interfacing service, and it is defined as a new component. The Composition Configuration defines how this component entity incorporates the delta "access Project definition". Same situation is for the PS Supplier information (design fragment 2). Using Design fragment Composition, the two design components can be composed. Note that the delta is not shown in the final design, they became the new behaviour (or functions) of relevant entities. In this case, "access Project definition" and "access Supplier definition" could be two java Servlets deployed in MAX server, they invoke PS API to obtain relevant information.

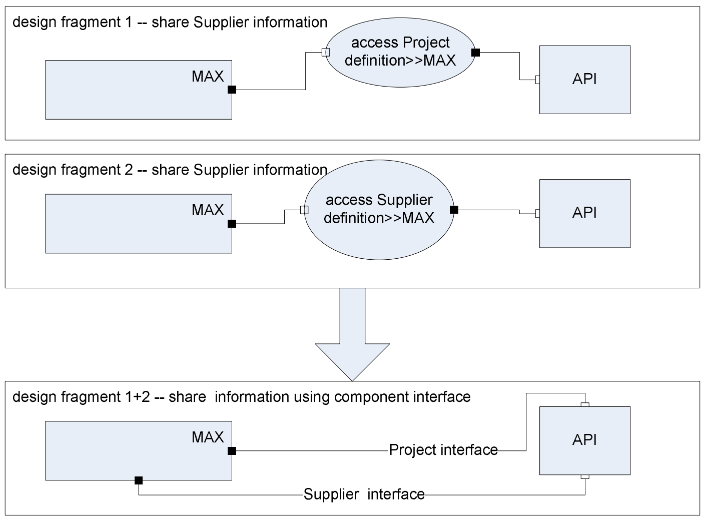

design fragment 1 -- share Supplier information
MAX
access Project definition>>MAX
API
design fragment 2 -- share Supplier information
MAX
access Supplier definition>>MAX
API
design fragment 1+2 -- share information using component interface
MAX
Project interface
API
Supplier interface

Figure 3 Archium -Design fragment composition

Design decisions will generate a number of additional requirements. In this case, once the use of MAX is decided, it will generate a series of design decisions issues, such as whether or not this MAX system be deployed as a centralized system. Archium will iterate step 2 and step 3 until all Problems have been addressed and all design fragments are composed.

4.3 Akerman and Tyree 's Ontology

This approach follows a five-step process to build up the ontology and reach the final architecture design by making architectural decision iteratively.

Step 1 -- Capturing stakeholder concerns
Business visions are articulated as a set of stakeholder concerns in this approach. In this approach, concerns are classified regarding different perspectives that architectural design should take into account. For example, the Concern in this case can be described as follows.

(class)Concern

(sub-class)Business Need:

(instance)Keep track of equipment maintenance activities

(instance)Monitor cost distribution

(instance)Reduce equipment down time

(sub-class)Change Case:

(instance)One to one mapping between equipment and financial asset in the future

(sub-class)Capability :

(instance)Work Order

(instance)Inventory

(sub-class)Quality:

(instance)Maintainability

(instance)Performance

(instance)Easy to use

(sub-class)Risk

(instance)ERP system upgrade and even change

(instance)System complexity keeps growing

Step 2 -- Analyse current architecture
IBM Architecture Description Standard is adopted by the Architecture Asset Segment to capture current Architecture with the concept of Systems, Subsystems, Components, Nodes, and Interfaces, etc .
In ontology tools, all these concepts are classes of type Architecture Asset. For SPC's current system architecture, PS system is an instance of class System. The three related sub-systems(FIN, SCM, HR) and 9 components(HR, GL, AR, AP, CC, PC, PR, PO, RCV) are the instances of corresponding class(Subsystem and Component). Interrelationships between components are recorded as the instance of call Interface, e.g., the RCV generating account entry for AP. The system's physical and deployment information are recorded in the class Data and Node, respectively. Architect can use this segment of ontology to analyse the gap between the current system and business needs described in the class Concern.

Step 3 -- Defining the target architecture
In this phase, the Architecture Decision Segment of Ontology is build up. This approach rely on Kunchten's framework for software architecture ontology[16].

After evaluating the Concerns in step 1, two alternatives (instances of Alternatives) are created, MAX and PS_EXT. Each alternatives has relationships of "helps", "neutral", "hurts" or "breaks" with relevant instances of Concerns. For examples, the relationship that PS_EXT breaks maintainability is an instance of class Concern-Alternative Relationship.

Architecture Decision are made by selecting an appropriate alternative, i.e. MAX is now an instance of class Architecture Decision. This decision will derive new issue, such as the deployment of the new system and system interfacing concerns. The Class "Implication" (sub-class of Concerns) is used to capture these new issues, and new decision loop begins. This loop will proceed to address all new Concerns and only stops when no more "Implication" is generated. Implication is used to generate architecture assets with the class Implication-Asset Relationship which takes the values "creates", "modifies", "uses" or "retires". For instance, the MAX decision "creates" MAX server and database as the instances of Node and Data, respectively.

Step 4 -- Conduct a gap analysis and produce the roadmap
The Roadmap Segment provides direction to migrate from the current system (described in step 2) to the target system (described in step 3).
Initiatives are a set of stakeholders concerns during the process of migration. In this case, that might be:

(1)step-by-step implementation

(2)site-by-site implementation if decentralized deployment is selected

Projects are:

(1)Equipment module implementation

(2)Work Order Module implementation

(3)Inventory Module implementation

(4)Master data interfacing

(5)Transaction data interfacing

(6)Roll-out

Step 5 -- Validate the architecture
Validating activities are usually as follows:

(1)How architecture meets stakeholder concerns identified in Step 1: Does the ERP plus MAX solution meet overall business requirement? Are all non-function requirements addressed appropriately?

(2)Review key architecture decisions

(3)Roadmap inspection: do the projects cover all the implications from the current system to the target system.

4.4 AREL

What AREL follows is an iterative process that starts from motivation and end with the design outcome, and design decisions with rationale connect them in between.

Step 1 -- Identify Motivation
Motivation comprises business requirements, the technical and organisational constraints on the architecture design, the assumptions which need to be observed. At the beginning phase of this case, the following AE elements can be found:

Motivation AE1: type(Functional Requirements): provide Site Maintenance Managements

Motivation AE2: type(Functional Requirements): Provide Inventory managements

Motivation AE3: type(Non Functional Requirements):System Performance

Motivation AE4: type(Non Functional Requirements):System Maintainability

Motivation AE5:type(Information System Environment):ERP-PS

Motivation AE6:type(Information System Environment):IS Infrastructure

Motivation AE7:type(Technology Environment):Open Integration Capability of PS

Motivation AE8:type(Business Environment):Current logistic business process

Step 2 -- Design decision & its justification
AREL supports three types of justifications, qualitative rationale, quantitative rationale and alternative architecture rationale. In this case, to determine whether a commercial MAX or extension of PS is adopted, the qualitative method could be used. The qualitative rationale can be in a textual format. The content could be business best practice, the evaluation of current PS system and its architecture, these document may be kept in a AR element (AR1) which may result in a design outcome Implementing MAX (AE9, which is the design outcome and also becomes a new motivation AE). Figure 4 show this fragment of design decision making and rationale capturing process.

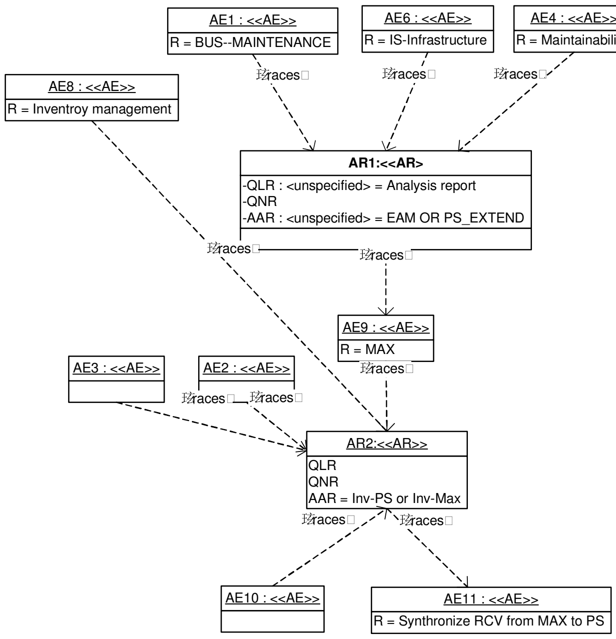


Figure 4 AREL- A Design Fragment

Step 3 -- Iterate step 1 and 2 until all architecture-level decision issues are addressed.
For this AE9, because the function overlap between the PS and MAX, system architect has to make a decision on which system will perform the inventory management.

Motivation AE2: type(Functional Requirements): Inventory managements

Motivation AE3: type(Non Functional Requirements):System Performance

Motivation AE8: type(Business Environment):Current logistic business process

Motivation AE9: type(Design Objects):Implementing MAX

Based on these motivations (note AE9 is a new one), a new design decisions on inventory management should be made, it is justified by the following AR:

AR2:

Decision Issue: Which system will the inventory management reside (Inv-PS or Inv-Max)

Alternatives: Implementing INV in PS and Implementing INV in MAX

Rationale: the evaluation of two Inventory modules based on current logistic business process; the evaluation of the impact of each alternatives on system performance.

This justification process may generate following two architecture elements.

AE10: type(Design Objects):implementing Inventory in MAX.

*AE11: type(*Design Objects):Synchronize RCV from MAX to PS.

AE10 and AE11 are the results of the decision on inventory management, these results set forth new design objectives for the rest of the architectural design. As shown in Figure 4, the relationships between AEs and ARs are represented by the UML stereotyped association <<ARtrace>> with which architect can trace the impact of AE (forward trace) and the root-cause of architecture design(backward trace). In this case, maintainers of the system can use AREL to retrieve the design rationale automatically to understand the reason and the constraints of synchronizing RCV (AE11).

4.5 AQUA

The best way to apply this approach is through a decision finding-evaluating-changing process on a candidate architecture, this decision-centric process will help to transform architecture to a better one with respect to system quality requirements.

At the planning phase of this project, based on a) industry best practice or b) iterative design deliberation though several formal or informal workshop, the project team has an initial architectural design as shown in Figure 5. In this design, all supply chain processes (i.e. PR, PO, RCV, INV) are migrated from PS to MAX, application messaging technology (PS-specific messag-based interfacing techniques) are used to transfer item/suppliers information from PS to MAX, MAX invokes PS Component Interface (requires an API installed in PS Web server) to access project information, and MAX Enterprise Adaptor(MEA) acts as the gateway connecting the two systems. The AQUA approach can be applied on this candidate architecture, to optimize it.

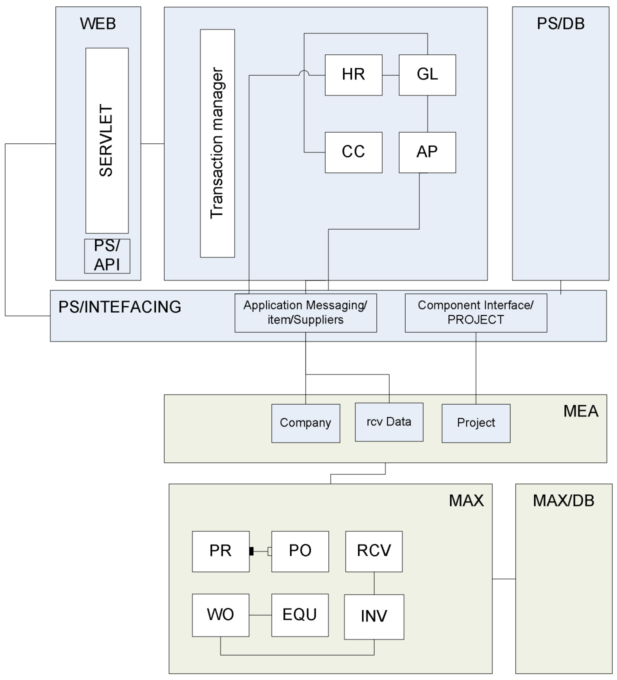


Figure 5 AQUA-Find Decision

Step 1 -- Find the architectural design decisions in current system:

D01:PS_EXT or MAX (current:Implement MAX)

D02:(current: Centralized Deployment)

D03:Generate Account payable Account Entry from MAX RCV

D04: Human Resources Information transfer from PS to MAX

D05:PR function migrated to MAX

D06:PO(current: PO function migrated to MAX)

D07: RCV performed in MAX instead in PS

D08:Suppliers information Synchronization

D09: ITEM definition Synchronization

D10: Project definition transfer from PS to Max

D11: Inventory in MAX

Step 2 -- Evaluate the decisions
We can draw a decision constraint graph(Upper part of Figure 6). As it shows, the decision d06 is an critical design decision and it will impact dramatically on other design decisions (d10,d08,d09,d05,d03,d07). If PO is migrated to MAX, then a lot of data synchronization will occur between the two systems, which will impact on the system performance and maintainability.

Step 3 -- Change the decision
 We consider another alternative for d06, that is PO function remains in PS, As described in the decision constraints graph, this decision will impact on (d10,d08,d09,d05,d07, d03). The architect can make the decision whether or not these decisions should change regarding d06's change. For example, d09 would be changed to "copy ITEM definition from PS to MAX". Because there is no need to develop complex synchronization algorithm which will be time-consuming, stakeholder's concern of quick implementation is more likely to be satisfied. The lower part of Figure 6 shows which part of architecture are influenced (depicted using darker color).

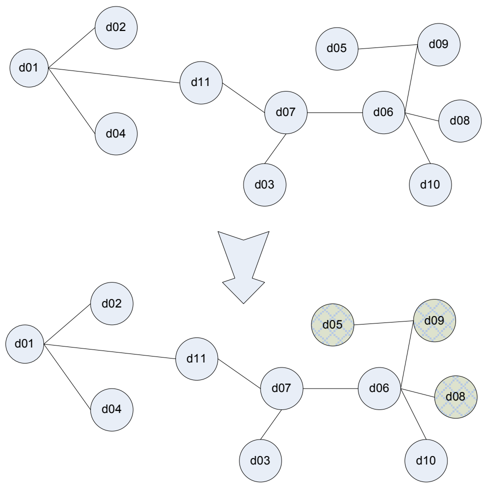


Figure 6 AQUA-Design Constraint Graph

From observing the above steps, it can be concluded that AQUA is more like an architecture evaluation and improvement methodology based on design decision finding and transforming. In [9], the author also compares AQUA with other architecture evaluation methods.

4.6 ArchDesigner

At the beginning of the system architecture design process, all of the design decision issues should be identified, design alternatives are developed and all quality attributes need to be collected. In ArchDesigner, the final architecture A=d1+d2+…+dn, where d is the design decision for each decision issue.

Step1 -- List all inter-dependent design decisions, along with their alternatives, and address the quality attributes desired by stakeholders. ArchDesigner supports decision making on inter-dependent issues that have competent impacts on software quality attributes. In this case, the alternatives for decision issues listed in Table 2 will impact on system quality attributes in different ways. For example, Implementing PR in MAX(PR-MAX) benefits system performance and “easy to use” but damage “easy to implementation”.

decision issue	Alternatives	Description
GEN	EAM-MAX	COTS adoption
	PS-EXT	Extend current ERP system(PS)
DPL	Centralized	All district business units share one installation of MAX which is centralized
	Decentralized	Each district business unit has its own MAX
INV	INV-PS	Manage Inventory using PS
	INV-MAX	Manage Inventory using MAX
PR	PR-PS	User submit purchase requisition in PS
	PR-MAX	User submit purchase requisition in MAX
	PR-MIX	User can submit purchase requisition in the system they prefer
PO	PO-remain	PO function and process remains in PS
	PO-MAX	Implementing PO in MAX to supplement
RCV	RCV-PS	Receive Materials in PS, sending receiving information to MAX for updating INV
	RCV-MAX	Receive Materials in MAX, sending receiving information to PS for generating accounting information.
INT	App-message	PS supported application messaging mechanism
	Com-Int	PS supported component interfacing mechanism

Table 2 list all the decision issues and their corresponding alternatives in ArchDesigner

Step 2 -- Value score computation
In this step, the value score of each alternative of each decision will be computed by weighting its Quality Support. ArchDesigner rely on the AHP method and compare the alternatives in a pair-wise fashion. Take the Decision issue PR as an example, architect will give the weightings as shown in Table 3.

Quality attribute	#1 in pair	#2 in pair	Quantitative weight[1]	Explanation
Easy to use	PR-MAX	PR-PS	1	PR only exist in one package will be easy for use
	PR-MAX	PR-MIX	9	

[1] (1 means equal important, 9 means extremely more important, 1/9 means extremely less important)

	PR-PS	PR-MIX	9	
Performance	PR-MAX	PR-PS	4	Currently PS is overloaded, if migrate some of its task to other system, the performance will be improved
	PR-MAX	PR-MIX	4	
	PR-PS	PR-MIX	1/9	
Easy to Impl	PR-MAX	PR-PS	1	Almost same effort needed for the implementation
	PR-MAX	PR-MIX	1	
	PR-PS	PR-MIX	1	

Table 3 PR's alternatives' quality support weight

Stakeholder's weighting on quality attributes uses the same pair-wise method, and each quality attribute will get an aggregate weight. Then the value score for each alternatives (PR-MAX, PR-PS, PR-MIX) of decision (PR) can be computed by above data. Generic AHP tool can be applied to do the calculation task.

Step 3 -- Global optimization
Although all alternatives' impact on quality attributes has been calculated, it is still not the time that PR decision can be made by selecting the alternative with the highest score. In our case, ArchDesigner will give a global optimization to the decision set listed in Table 2, not on a single decision basis.

ArchDesigner will proceed with all design decisions, and value the score computation with each decision. According to the different extent of influence on architecture for each decision, the score of each decision will be submitted to a normalization process before the optimization. For example in our case, the PO decision may have significant influence on the whole architecture, so it will get a relatively higher weighting compared to other decisions.

Using Integer Programming, global optimization is used to maximize the accumulative score, associating other constraints formulation, such as time and cost. The combination of alternatives with the highest value score will be the outcome of ArchDesigner.

4.7 BBN-based Design Alternative Selection

Similar to ArchDesigner, at the beginning of the BBN-based Design Alternative Selection approach, the architect has to identify all decision issues and quality attributes. This Approach uses a DAG (Directed Acyclic Graph) to explicitly model decisions, quality attributes, and their relationships (Figure 7 shows parts of this BBN network as an illustration). In this graph, we can see the relationships between decisions are managed, for example, if PS_EXT is selected, the useMax solution for Inventory will not be selected.

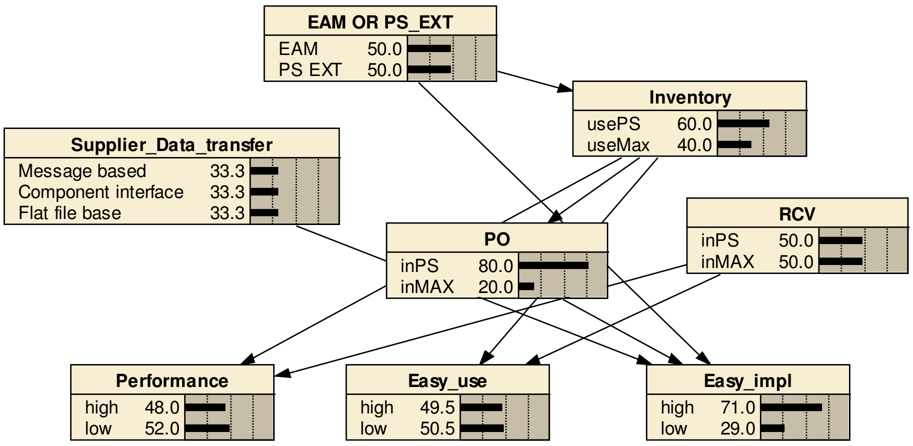
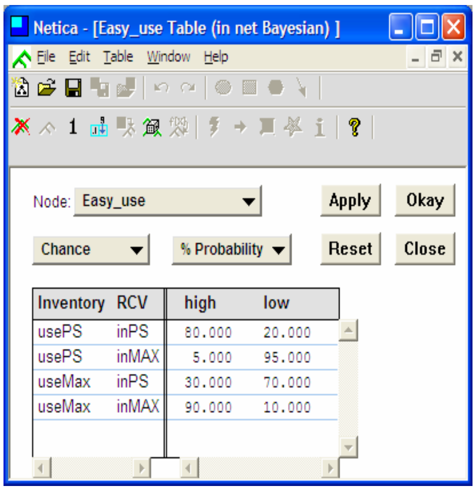


Figure 7 Bayesian Belief Network -- decisions impact on Quality Attributes

All these probabilities are based on architect's belief in how much a given decision (or a set of combinations) influence a quality attribute. Figure 8 shows each decision node is attached with a CPT(conditional probability table) to assign the probability value. When all probabilities have been determined, Bayesian network tools (Necktie, for example) can be use to calculate the value of each node. Within this network, architect can propagate decisions over the network and update the probability of each node and see the different result in certain quality attributes and find a desired decision. As shown in Figure 9, if MAX is selected, the architect can investigate the probability change of each quality attribute (Performance, Easy_use, Easy_impl) to see the impact of this decision.


Netica - [Easy_use Table (in net Bayesian)]
File Edit Table Window Help
Apply
Okay
Reset
Close

Node: Easy_use

Chance | % Probability

Inventory	RCV	high	low
usePS	inPS	80.000	20.000
usePS	inMAX	5.000	95.000
useMax	inPS	30.000	70.000
useMax	inMAX	90.000	10.000

Figure 8 Bayesian approach - CPT table

Although the nodes in Bayesian network could be numerous, in our case we find that if one attribute is influenced by more than 3 decisions, it will be difficult for domain expert to determine the probability value(in case of 4 decisions, the rows of CPT will be at least 16).

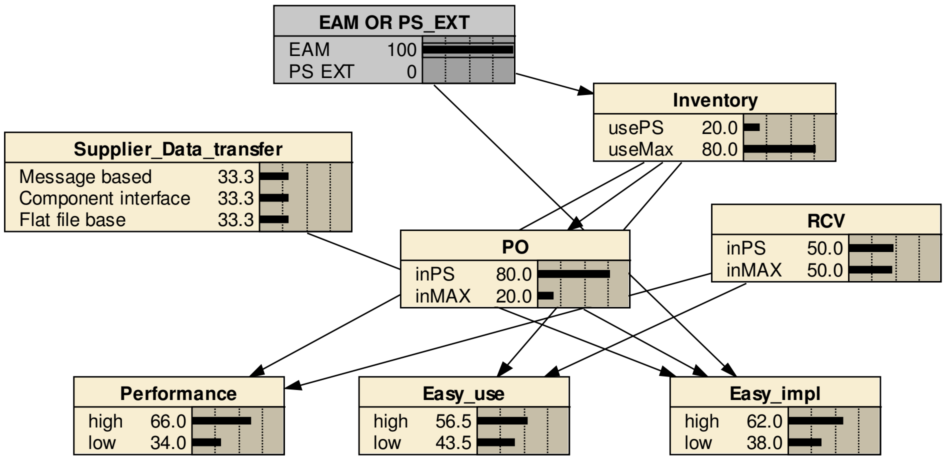


Figure 9 Bayesian approach- the updated Bayesian Network after selecting MAX

4.8 Automated Solution Synthesis Approach

The input of this approach is software requirements (FRs and NFRs), the output is the decided software architecture, recorded decisions and rationale. We follow the four steps suggested by this approach.

Step1 --Issue Eliciting

This approach focus on architecturally significant issues, and it adds value to the decision process only when these issues are inter-dependent, which means the solutions to these issues are intertwined, and has the relationship like INCLUSIVE, GENERALIZED, and CONFILICTIVE. Based on the FRs and NFRs of this case, we select six issues that are suitable to illustrate this approach.

Issue 1(PR): Purchase Requisition business process

Issue 2(RCV): Where is RCV processed?

Issue 3(INT_Supplier): The technology of Interfacing, message based or Component (API) based For Suppliers Information

Issue 4(DPL): The system deployment architecture

Issue 5(INT_Proj): The technology of Interfacing, message based or Component (API) based For Projects Information

Issue 6 (PO): How Purchase Order Process performed in the two systems?

Step 2 -- Solution Exploiting

For each issue listed above, find all candidate solutions, the pros and cons of each issue solution are also specified, see Table 4.

Issue	Issue Solution	Description, Pros and Cons
PR	PR-PS	User input PR in PS. Pros: All PRs are recorded in one system Cons: Users need to use two system; System keeps to be slow
	PR-MAX	User input PR in MAX. Pros: Burdens of PS can be reduced Cons: some users need to use two system; ITEMS data need to be synchronized

	PR-MIX	Maintenance PR are input in MAX, others are input in PS Pros: Burdens of PS can be reduced; no users need to use two system Cons: Both the users and system are need to be educated to discriminate maintenance PR and non-maintenance PR
RCV	RCV-PS	Receiving material in PS Pros: it is currently running in PS Cons: transaction data need to be transferred to MAX synchronously to update inventory, high risk
	RCV-MAX	Receiving material in MAX Pros: only summarized accounting information transferred, low risk Cons: new implementation activity needed
INT_Supplier	App-message	Supplier information are transferred from PS to MAX using message subscribe/publish Pros: easy to implement with the help of MAX MEA Cons: not real time interfacing
	Com-Int	Supplier information are transferred from PS to MAX using component interface Pros: real time Cons: Customization in MAX side
DPL	Centralized	MAX is installed in the head office and shared by all business units Pros: easy to plan interfacing with PS Cons: hard to deal with differences of business process among business units.
	Decentralized	Each business unit has its own MAX installation Pros: easy to deal with diversity. Cons: interfaces are difficult to implement and monitor;
INT_Project	App-message	Project information are transferred from PS to MAX using message subscribe/publish Pros: easy to implement with the help of MAX MEA Cons: not real time interfacing
	Com-Int	Project information are transferred from PS to MAX using component interface Pros: real time Cons: Customization in MAX side
PO	PO-remain	PO function remains in PS Pros: minimum system modification Cons: The whole supply chain are separated in two system
	PO-MAX	PO function are migrated from PS to MAX Pros: maintain an integrated supply chain in one system Cons: more customization efforts in MAX project; too much data need to be transferred

Table 4 Solution Exploiting for each Issue

Step 3 -- Solution synthesizing
Automated Solution Synthesis will try to find all possible combination of issue solutions and eliminate the conflicted combination.

First, relationships between any two solutions are identified. Table 5 shows the relations between Issue Solutions (solutions that are independent of each other are not included in this table). INC, GEN and CON represent the INCLUSIVE, GENERALIZED, CONFILICTIVE relationship between solutions respectively. For example, if Decentralized deployment is selected, then only application messaging based interfacing method can be employed because the network latency issue render the other alternatives unviable. This restriction is recorded in this table as a Generalized relationship. Another example is for the INCLUSIVE relationship between the PR and PO decision issue, if the functionality of Purchase Order is migrated to MAX, the Purchase Requisition process will be migrated to MAX also to avoid purchasing requisition data transferring from PS to MAX.

Relation	Issue Solution
INC	(PO_MAX, PR_MAX)
GEN	(DPL:Decentralized, INT_Supplier:App-messaging), (DPL:Decentralized, INT_Project:App-messaging)
CON	(RCV-PS, INT_Project:App-messaing), (RCV-PS, INT_Supplier:App-messaing)

Table 5 Relations between issue solutions

Second, find all solution combinations using a recursive algorithm, this algorithm will build up a solution combination tree. In this case, for the six decision issues, there should be (3*2*2*2*2*2=96) candidates, but with the help of this algorithm, most candidates are eliminated automatically because of the restrictions defined in table 5 (24 candidates are left).

Third, merge the solutions into candidate architectures for each of the remaining combinations.

Fourth, make a final decision on architecture
In this step, these candidate architectures are evaluated based on the summed pros and cons and other additional evaluations, one candidate is selected as the final architecture. Each solution is associated with the pros and cons as design rationale to assist decision making.

4.9 PAKME

PAKME captures information with its various knowledge acquisition templates. With this case, the following architectural artefacts could be recorded in the knowledge base:

Project information: This enterprise system evolution project “contains” a series of architectural decisions (i.e. PS_EXT or EAM, Centralized or Decentralized Deployment). Project also “contains” architecturally significant requirements (ASR), for examples maintainability and easy of implementation etc.

User Group: PAKME will classify all system users into several groups: maintenance managers, workers, purchase supervisors and inventory managers, etc. Each user will also be identified in PAKME and is linked with user group.

Scenario: PAKME captures both general and contextual scenarios. General scenario may be “systems should allow users to cancel operations”, example contextual scenario is “in planning a maintenance task, the maintenance manager has to check the warehouse to make sure materials are available”.

Patterns: general design patterns are provided by PAKME. For this specific case of system integration, architect may refer relevant patterns such as Observer, Proxy, Publish/Subscribe, etc. Integration techniques provided by PS and MAX, including Component Interface, Business Interlink, Application Messaging, Max Enterprise Application can also be input into PAKME and linked with certain decisions.

Architectural Decisions and design options: architectural decisions can be described at different levels of granularity. For example, "PS_EXT or EAM" is a high level decision, while "suppliers information integration" is a fine-grained level decision on the issue of integration mechanism. Both decisions and design options have associated design rationale. Attributes of Architecture decisions include: concrete scenario it addresses, design options, design history, relevant documents, relationships with other decision.

Findings: PAKME can record evaluation results during the architecting process, and they are captured in the "finding" template. "Finding" could be a risk or a non_risk description such as " A risk arises when PS releases a new patch", this "finding" is associated with the architecture decision "PS_EXT" and concrete scenario "applying patches from software vendors".

PAKME does not provide system modelling techniques or direct decision support techniques. However, it works in parallel with these processes and captures various architecture knowledge.

4.10 ADDSS

ADDSS captures design decisions under the iterative development process. Initially in this case, the following architectural knowledge is input into the database:

System: Current system (PS) description and its thumbnail figures of architecture.

Requirements: both the functional and non-functional requirements are recorded.

Stakeholders: all stakeholders of current system and future system

Then the architecting process begins. ADDSS uses "Iteration" to keep the history of design process, and this historical list of design decisions and the interim architectural diagrams can be used by the stakeholders to trace the rationale.

The decisions capture the following information:

Category of the decision: the category discriminates between the "main", "alternative" and "derived" decisions. Example of main decision is "PS_EXT or EAM", and the "Centralized or Decentralized deployment" is a "derived" decision of this main decision, which records a parent-son relationship.

Status of decision: Each design decision is assigned a status attribute (Pending, Rejected, Approved, Obsolete, etc).

Requirements: The requirements addressed by this design decision.

Iteration: the Iteration in which this decision is made.

Style, Pattern: The Style, and Pattern used by this decision.

5. An Analysis

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

5. 1 Decision-centric architecture knowledge modelling

The elements in the Core model are categorized into three spaces to simplify our comparison and analysis. The problem space includes stakeholder, role, concern, decision topic. The decision space includes alternative, ranking, decision, architectural design decisions. The solution space includes architectural design, language, and artefacts.

Perspective 1.1: Description of Problem Space

In the problem space, Archium uses Problem with its Cause and Motivation to identify decision topic (see table 1). Archium also models requirements and integrates requirements model with architectural design decision model and the architectural model [4]. Requirements are categorized into a parent-child hierarchy, Scenarios are used to illustrate Requirements, and these two concepts express the "Concern" of the Core Model. The Requirements model is described with an ADL based language and is simple to access. Archium supports the tracing from requirements to architectural decisions and design fragments. As such, architects can evaluate the impact of a change before applying it. However, Archium has to re-compile to accommodate the change. Archium does not elicit ASRs from general requirements.

Akerman and Tyree's Ontology classifies Concerns into five different sub-classes through which "Concerns" are organised and software architect can evaluate alternatives from different angles. Akerman and Tyree's ontology does not employ a "requirement-requirement relationship" class and the relationships between the stakeholder concerns are not included in the ontology. This ontology uses Change Case to predict future change of requirements.

AREL uses Architectural Elements (<<AE>>) to encapsulate the motivational reasons, which can be a requirement, a goal, an assumption, a constraint or a design object. All these motivational reasons are considered as "Concerns" and explicitly represented as inputs to the design decisions.

ADDSS, AQUA and Automated Synthesis Approach simplify the problem space as Requirements. Those three approaches do not differentiate functional and non-functional requirements. PAKME models ASR and Qualify Factors and use Scenarios to describe these two factors.

ArchDesigner and the BBN-based Design Alternative Selection approach, however, only document the system quality attributes. Although they are "Concerns" of design reasoning, the entire problem space is not fully covered.

Both the IEEE 1471-2000 standard and the Core Model define Stakeholder in the conceptual model. Although Stakeholders initiate Concerns, not all approaches take stakeholders into account. ArchDesigner evaluates the different requirements of different stakeholders on quality attributes. Archium, AREL, PAKME and ADDSS relate Stakeholders with requirements. Table 6 summarize the above analysis.

Approaches	Modelling	Modelling	Surpporting Requirements

	Stakeholders	Concerns	Change
Archium	stakeholder is modelled in the requirements model of Archium, where requirements are “desired by” stakeholder	Problem with its motivation and cause is the center of problem space; Requirements are presented in a category hierarchy; Scenarios are used to realizes requirements	support requirement impact analysis
Akerman and Tyree Ontology	n/a	*concern* segment of ontology (concern has five subclasses)	use *change case* to predict future change
AREL	Stakeholder is associated with each requirement	Motivational reasons can be a requirement, a goal, an assumption, a constraint or a design object	support forward traceability for problem analysis.
BBN-based Design Alternative Selection	n/a	capture quality attributes but not the other types of design concerns	n/a
AQUA	n/a	capture requirements, risks, design constraints	add/delete/modify nodes in decision constrains graph, but manually.
ArchDesigner	stakeholder who have relative preference on quality attributes	quality attributes	n/a
Automated Synthesis	n/a	FR and NFR issues (architecturally significant requirements)	n/a
PAKME	user group/user user identifies ASR and scenario	ASR, quality factors and scenario (ASR and quality factors are described by scenarios)	n/a
ADDSS	stakeholder	FR and NFRS	n/a

Table 6: The problem space description of each approach

Perspective 1.2: Descriptions of design decisions

We have found that all approaches document design decisions explicitly in the architectural designs but they do it in different ways. Some of the studied approaches represent design decisions with a conceptual decision model whilst the others informally organize design decisions. We analyse the representation of design decisions in terms of the three characteristic segments summarized in the ontology [16] of architectural design decision: the classification, attributes and the various relationships.

Most approaches do not classify design decisions except Automated Solution Synthesis and ADDSS. Automated Solution Synthesis has two levels of decisions, solution-level decision and architecture-level decision. ADDSS(ver2) classifies design decisions as "main", "alternative" and "derived", alternative decisions are design options which is missed in ADDSS version 1, main and derived decisions are used to determine the granularity of decisions. Neither of them comply with the taxonomy proposed in [16].

Attributes of design decisions may include Description (Epitome), Rationale, Scope, State, History, Cost and Risk[16]. However, those decision-centric approaches diverge in defining decision attributes (see table 7).

All studied approaches relate design decisions with upstream artefacts (in problem space) and/or downstream artefacts (in solution space). AREL uses <<ARtrace>> to trace the relationship between AE (motivations are Concerns) and AR (design decision with its rationale). In PAKME, Design decisions are related to scenarios rather than requirements. An architecture decision is "identified" by scenarios which "describe" architecturally significant requirements and quality factors. ADDSS, AQUA and the Automated Synthesis Approach model the relationship in a similar way ("satisfy" in AQUA, "examines" in ADDSS and "address" in Automated Synthesis Approach) between decisions and upstream artefacts (requirements). ArchDesigner and BBN based Solution Selection approach, however, document the quantitative impact of design options on quality attributes.

As architectural design is a set of design decisions, the relationships between these decisions provide a mapping of how they influence each other in the design process or in the results of the design. However, we have found that not all decision approaches address the relationships between design decisions.

PAKME and Akerman's Ontology explicitly maintain various dependency relationships, the dependency types comply with the relationships defined in [16], such as "constrains", "forbids", "enables", "subsumes", "conflict with", "comprises".

ADDSS uses decision tree to define relationships between decisions. The nodes of tree (or network) represent design decisions and each node stores the information about the requirements that affect a particular decision. The current release of ADDSS only defines the "comprises" relationship [16] between decisions and uses a tree to represent the hierarchy. Generally, the depth of this design decision tree is the number of decision iteration, and the deeper is a decision in this tree, the higher level of granularity it contributes to architectural design. User has the ability to relate decisions and establish dependencies among them. In this case, the tree structure becomes a network of decisions and the tree structure will be *deformed* [26].

As shown in the illustrating case (Figure 6), AQUA has a similar decision graph (Decision Constraint Graph) as ADDSS's decision tree. But these two are different in two aspects. One is the node in the graph. The node of Decision Constraint Graph is "decision variable", a concept in AQUA meaning decision topic or decision issue. Each node may associate several design options or alternatives. The node of decision tree in ADDSS is the design decision itself. Another difference is the edges in the graph. Edge in Decision Constraint Graph represents constraint relationship between two decision variables. For example in the illustrative case, the system that the PO function resides in (decision variable d06) constrains how the ITEM synchronizes between the two systems (decision variable d09). The edge in decision tree of ADDSS represents the parent-child or “comprises” relationship between two design decisions. An example is the decision MAX, which comprises a series of more specific design decisions.

The BBN-based Design Alternatives Selection approach also uses graphical model to capture the relationships among design decisions and quality attributes. From the perspective of design decision relationship, the difference between this graph and Decision Constraint Graph (in AQUA) is that 1) BBN is a quantified model using subjective probability to quantify the relationship while AQUA is qualified; 2) design alternatives are modelled as “state” of each node in BBN and must be predetermined, but in AQUA alternatives description can be more flexible, architect can nominate a new solution even at the architecture evaluation phase.

Archium provides a visualized graph of design decisions which records the functional dependencies between architectural decisions. AREL provides traceability to find the relationship. For example, backward tracing of one architecture entities (Architecture Elements in AREL) gets a chain of design decisions (with rationales) which are related because they lead to a common architecture entity. Hence, these two approaches records the “dependency” relationships which correspond to the “constrains”, “comprises” and “overrides” relationships defined in [16].

The ArchDesigner and Automated Synthesis approaches, on the other hand, document the dependency relationships between alternatives (solutions) of each decision, for instance, one solution may “conflict” with a solution of another decision issue. These kinds of relationship are used by their algorithm to select or discard certain alternatives.

In the case study, we find that the knowledge vaporization problem is the target of all approaches and they explicitly document design rationale, this facilitate the reusing of architectural knowledge. However, the issue of dealing with design decisions that are cross-cutting and intertwined are not addressed by all the approaches.

Table 7 summarize the above analysis.

Approaches	**Design decision categories**	**Attributes in design decisions**	**Relationships between design decisions and upstream artefacts**	**Relationships between design decisions**	**Relationships between design decisions and downstream artefacts**	**Problem Resolution**
Archium	No	Description, design rules, design constraints, consequences , pros and cons	Design decisions “creates”, “obsoletes”, “Realizes”, “Uses” “Threatens” requirements/scenarios	“Dependency” relationship	Design fragment	KV RULE
Ontology	No	Descriptions	design decision	Use (Decision-	implication	KV

			"makes", "helps", "neutral", "hurts" and "breaks" a specific concern	decision Relationship) to record various relationship		CC-INTER
AREL	No	Architecture rationale encapsulates issues/questions, arguments tradeoffs, alternatives	Motivation(AE) acts as an input to a decision. It "motivates/constrains" a decision	Linked by architecture elements	AE as design outcome	KV RULE
Bayesian	No	No	Decision's quantified impact on quality attributes	Present in DAG using nodes and edge to record the "impact" relationship	no	CC-INTER
AQUA	No	Decision is composed of a pair of decision variable and decision value	Decisions "satisfy" Requirements Decisions are "part of" Risk	Use Decision constraint graph to record the relationship between decision variables	no	KV CC-INTER RULE
ArchDesigner	No	No	Alternatives of specific design decision have "different impact" on quality attributes	no	no	KV CC-INTER(partial)
Automated Solution Synthesis	Issue decision and Architecture Decision	Descriptions , pros and cons (for issue solution)	Issue Decision (by adopting a issue solution) "solve" an Issue	Record relation between solutions using inclusive, generalized, conflictive and independent	Instantiated solution	KV CC_INTER
PAKME	No	Compatible with [16]	Architectural decision is "identified" by scenario	Record various relationships, "constrains", "Excludes"	no	KV Rule
ADDSS	Main, Derived,	Descriptions, Status,	Stakeholder examines FR	Use decision tree (network)	Hyperlink to imported design	KV Rule

	Alternative	category	and NFRS, then takes a Decision	to record the “comprise” relationship	diagrams	

Table 7: Description of decision space

Perspective 1.3: Description of solution space

In Archium, a design decision results in a design fragment which is described by components and connectors to form an overall architecture (component and connector view).

Akerman & Tyree’s Ontology uses the “implication” class as the outcome of design decision, the implication can be related with one or more “Architecture Asset”. Akerman & Tyree’s Ontology adopts IBM Architecture Description Standard to represent architecture and it supports generating standard architecture views with plug-ins.

AREL uses AE (Architectural Elements) to model design outcome. Design decisions are embedded in AR (Architectural Rationale) and they are connected to architectural elements (AE) directly with ARtrace. AREL also classifies AE by viewpoints (business, data, applications and technology), and consequently the software architecture can be observed and investigated from these different perspectives.

The Automated Synthesis Solution approach uses “issue solution” (design fragment) to instantiate an issue decision. Issue solution is modelled with components, connectors and structures, is also the composited candidate architecture. Although this approach records the relationship between decisions and design fragments during the solution instantiating phase, the relationship between decision and architectural design is broken because the issue solutions need to be manually composited to candidate architecture.

AQUA, the BBN-based Alternative Selection and ArchDesigner concentrate on the decision making process(AQUA may use traditional software architecture description to evaluate system and find design decision issues). They do not describe architecture and have no conceptual counterparts with the Artefacts in the Core Model (Artefacts). The output of these three approaches is just a set of design decisions.

ADDSS doesn’t define how software architecture is represented. Users need to upload figures from external files representing architectures. ADDSS provides multi-perspective architectural view definition. However, these views are uploaded from external architecture design tools, and ADDSS just documents and organizes them.

The current version of PAKME does not support software architecture representation.

The solution space comparison of each approach is shown in Table 8.

Approaches	Solution Space Support	Viewpoint/View Support
Archium	Design fragments presented by components and connectors	No support to viewpoint/view
Ontology	Implications	No support to viewpoint/view

AREL	Architectural Elements represented by UML stereotype	Support Viewpoint
Bayesian	A set of design decisions	No support to viewpoint/view
AQUA	a set of design decisions	No support to viewpoint/view
ArchDesigner	a set of decisions	No support to viewpoint/view
Automated Synthesis	issue solution, architecture candidate and architecture design	No support to viewpoint/view
PAKME	no	No support to viewpoint/view
ADDSS	Architecture view loaded from modelling tools	Organize view according to stakeholder's interests

Table 8: the solution space

When we examine the approaches from the architecture knowledge modeling perspective (i.e. Perspective 1), they 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, Bayesian Network based Alternatives Selection Method 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 is an important aspect of architectural description [1], only ADDSS and AREL provide different views or viewpoint support in representing architectures.

5.2 Decision Making Techniques

Based on the case study in section 4, we analyse specific decision making steps and techniques utilized in each approach, and try to map these steps with the general architectural design activities [18].

Perspective 2.1: Specific steps for decision making

Different processes and techniques are used by different approaches to support decision making. The BBN-based Alternative Selection Approach and ArchDesigner employ quantitative methods while others use qualitative method.

Archium, AREL and Arkman's Ontology have similar decision making steps that include the following [27]: 1) identify and define the problem; 2) determine the set of alternative solutions; 3) determine the criterion or criteria that will be used to evaluate the alternatives; 4) evaluate the alternatives; 5) choose an alternative. These three approaches iterate the decision making process till all architectural design issues are covered. This iterative process exactly matches the "decision loop" in the Core Model.

The BBN-based Design Alternative Selection approach and ArchDesigner focus on alternative ranking. Both of them quantify the overall architecture value by measuring alternatives' impact on quality attributes. The BBN-based Alternative Selection approach addresses the inter-dependency relationship between decisions under the situation that one design decision may impact other decisions, as in our case, the (MAX, PS_EXT) decision's impact on Inventory decision is defined in the BBN network; ArchDesigner's objective function is only an accumulative value score of each decisions, missed the inter-dependency relationship among design decisions(although in its optimization process, conflict alternatives from different decisions will not be put together). The strong point of ArchDesigner is that the project cost and time constraints are taken into account in the optimization formulation, while the BBN-based Alternative Selection Approach only evaluates the predetermined quality attributes. Another weak point of the BBN-based Alternative Selection approach is its lack of prioritizing on quality attributes and design decisions. Take our case as an example, the easy-to-use requirement may have more value to system users than easy-to-implementation, and the PR and PO design decisions have more significant influence on the system than interfacing implementation which is at the technical level and is transparent to system users. ArchDesigner uses weighted quality attributes and normalization to measure these two priorities.

The Automated Solution Synthesis Approach provides an automation algorithm to generate architecture candidates from the issue solutions and lessen the burden of architect in exploring 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, e.g., the (PO, RCV, DPL, INT) decisions in our case. However, in a large system where many more decision issues exist, the automation algorithm will generate a huge number of candidates, reducing the effectiveness of this approach.

Compared to other approaches, AQUA provides a totally different decision making process, which is finding decision, evaluating the decisions and then changing the 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, and how to maintain the decision constraints graph in large software system remains an issue.

PAKME and ADDSS document design decisions in alignment with their decision making processes. Both of them provide a repository of architectural styles and design patterns that can be referred by the architect when making decisions, both of them provide auxiliary support to decision making.

Perspective 2.2: Decision making level

Archium, ADDSS, PAKME, Akerman's Ontology and AREL work at the localized level, and they adopt an iterative decision making process as what we have seen in the case study.
The BBN-based approach and ArchDesigner work at cross-cutting design level. They intend to optimize the outcome of the design under the circumstances that design decisions are cross-cutting and intertwined. AQUA and the Automated Solution Synthesis Approach also address the cross-cutting issue, although they do not provide optimization to system designs. They suggest that software architecture should be evaluated as a whole, not only at the localized level.

Perspective 2.3: Supports on general design activities

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.

The summary of the three sub-perspectives of Decision making techniques and the limitation analysis are show in Table 9.

Approaches	**Decision Support Techniques**			
	steps for decision making	**decision making level**	**Limitations**	**Supports on general design activities**
Archium	No specific technique	Localized level	complicated and specific technical environments (ADL, Java, etc) are hard for architects to design and document AK.	AA AS AE
Ontology	No specific technique	Localized level	Difficult to generate views	AA,AS,AE
AREL	Use <<AR>> to accommodate qualitative, quantitative techniques	Localized level	Evolution trace is manual procedure	AA AS AE
Bayesian	Bayesian network	Cross-cutting design decision level	Not adaptable to requirement change	AA AS
AQUA	Architecture evaluation and impact analysis support for transformation	Cross-cutting design decision level	missed the relationship between design decision and software architecture without tool support	AA AE
ArchDesigner	Quality Objective function optimization	Cross-cutting design decision level	Not adaptive to requirement change	AA AS
Automated Synthesis	Automated solution synthesizing using a solution combining recursive algorithm	Cross-cutting design decision level Localized level	Not adaptive to requirement change	AS,part of AE and AA
PAKME	Provide reference to design patterns and styles	Localized level	Incur project overhead during the knowledge capture process	AA, AS, AE
ADDSS	Provide reference to design patterns	Localized level	Incur project overhead during the knowledge capture process	AA, AS, AE

Table 9: The Decision Support Techniques of each approach

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 Bayesian Network based Alternatives Selection Method 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.

5.3 Design decision rationale.

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 this solution is adopted or discarded and justifying other design decisions.

AREL incorporates the architecture rationale conceptual model into the design process which makes design rationale (AR) an significant element in the final software architecture.

Akerman's Ontology is weak in capturing design rationale. However, the wide range of relationship classes defined between architecture elements can be the clues to understand why an alternative is selected or not.

The BBN-based approach and ArchDesigner do not explicitly take design rationale into account, but the reasoning process, algorithm and quantitative evaluation constitute the design rationale. In the BBN-based approach, the Bayesian network shows an overall influence of design decisions on quality attributes, and it can be reused when designing similar systems in the same domain (see Figure 9). Once the Bayesian network is built up, it also can be used for the purpose of system maintenance and evolution. In ArchDesigner, the comparative contribution of alternatives to quality attributes and the weighting of quality attributes are kept as rationale (as Table 3 shows).

AQUA uses Decision Constraints Graph to support decision making and keeps track of decision changes, as illustrated in Figure 8. Automated Solution Synthesis integrates rationale into the architecture design process, and the reason why an issue solution is adopted or discarded is also recorded automatically.

In PAKME, the design rationale of a design option is based on "practitioners" opinions about rationale reported in [20]. Rationale of design decision describes the reasons, justifications and tradeoffs leading to this decision. They are documented in a PostgreSQL database. In ADDSS, design rationale is also captured using templates and input into the system manually. Design rationale is stored with design decisions in a knowledge repository. In addition, the decision tree (network) and the attributes of decision link provide support for architecture evaluation and evolution. However, in its design decision models, rationale is not a separate concept and ADDSS does not regulate the content or template of rationale.

Perspective 3.2: Using and retrieving design rationale

Archium provides a visualization tool to support retrieving architectural decision and its divers and rationales. In AREL, ARtrace enables architect to trace 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, Akerman's Ontology uses standard query language to retrieve information. The BBN based solution selection approach and the ArchDesigner approach do not provide retrieving support. AQUA currently has no tool support for its decision model and constraints graph, and any retrieving activity has to be performed manually. PAKME and ADDSS use hyperlink associated with design decisions or design solutions to access design rationale.

The two perspective of rationale of each approach is shown in table.

Approaches	**Design rationale**	
	Capturing and Documenting method	**Retrieving Rationale**
Archium	Capturing: Template-based evaluation; Pros and cons of solutions	Visualization of architectural decision by a dependency graph
Ontology	Template-based evaluation Various relationship managed by ontology tool	Standard query language provided by ontology tool
AREL	Incorporate architecture rationale conceptual model and elements linkage into design process quantitative , qualitative and alternatives rationale encapsulate in <<AR>>	ARtrace and AREL tool support for traceability
Bayesian	Reasoning process; Conditional probability table	Manual
AQUA	Evaluating the decisions in Decision constraint graph	Manual
ArchDesigner	Reasoning process ;AHP pair-wise comparison matrix, measuring the impact of alternatives and stakeholder's priority on quality attributes	Manual
Automated Synthesis	Solution synthesis algorithm; Pros and cons of solutions	Manual
PAKME	Template and manually input	Hyperlink provided in decision/solution template
ADDSS	Template and manually input	Hyperlink

Table 10: Design Rationale in each Approach

We have observed that, compared to rationale capture, design rationale retrieval are neglected in some approaches (i.e. Bayesian Network based Alternatives Selection Method, 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.

6. Findings

Using the three key perspectives to analyse the nine decision-centric design methods, we have identified a number of areas where general improvements could be made:

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 [28]. 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 intertwined 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 Bayesian Network based Alternatives Selection Method 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 Bayesian Network based Alternatives Selection Method 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 Bayesian Network based Alternatives Selection Method 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 analyse 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 rationales) 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 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.

7. Conclusions

In this report, we have analysed 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. Through the applications of these approaches, we have observed that although they provide different techniques to perform or represent design reasoning, there are key assumptions that they rely on. 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 that design reasoning is based on. In particular, we have 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 .

References

[1] I. S. 1471-2000, "IEEE Recommended Practice for Architectural Description of Software-Intensive Systems," 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," *1st ACM Workshop on SHaring ARchitectural Knowledge (SHARK),* 2006.

[13] 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.

[14] H. Obbink, P. Kruchten, W. Kozaczynski, R. Hilliard, R. Kazman, and A. Ran, "Report on Software Architecture review and assessment (SARA)," in *http://philippe.kruchten.com/architecture/SARAv1.pdf*, 2002.

[15] B. Nuseibeh, "Weaving Together Requirements and Architectures," *Software management,* vol. March, pp. 115-117, 2001.

[16] P. Kruchten, "An ontology of architectural design decisions in software-intensive systems," *Intern. Workshop on Software Variability Management,* 2004.

[17] L. Bass, P. Clements, and R. Kazman, "Software Architecture in Practice," 2003.

[18] 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.

[19] A. H. Dutoit and B. Paech, "Rationale Management in Software Engineering: Concepts and Techniques," pp. 1-48, 2006.

[20] 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.

[21] 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.

[22] 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.

[23] 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.

[24] J. C. Duen?as and R. Capilla, "The decision view of software architecture," in *Lecture Notes in Computer Science*, 2005, pp. 222-230.

[25] P. B. Kruchten, "4+1 view model of architecture," *IEEE Software,* vol. 12, pp. 42-50, 1995.
[26] R. Capilla, "A Web-based Tool for Managing Architecture Design Decisions," 2006.
[27] D. R. Anderson, D. J. Sweeney, T. A. Williams, and K. Martin, "An Introduction to Management Science, Quantitative Approaches to Decision Making," 2008.
[28] 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.