ISR Institute for Software Research
University of California, Irvine

A Survey of Software Architecture Decision-Making Techniques

Lihua Xu
University of California, Irvine
lihuax@ics.uci.edu

Debra J. Richardson
University of California, Irvine
djr@ics.uci.edu

Hadar Ziv
University of California, Irvine
ziv@ics.uci.edu

December 2007

ISR Technical Report # UCI-ISR-07-10

Institute for Software Research
ICS2 217
University of California, Irvine
Irvine, CA 92697-3455
www.isr.uci.edu

A Survey of Software Architecture Decision-Making Techniques

Lihua Xu, Debra J. Richardson, and Hadar Ziv
Institute for Software Research
University of California, Irvine
Irvine, CA 92697-3425
{lihuax, djr, ziv} @ics.uci.edu

ISR Technical Report # UCI-ISR-07-10

December 2007

Abstract: Software quality attributes describe both the specific criteria related to how the system is built (e.g. cost, development time), and qualitative constraints on various attributes of functions or services that the system should provide (e.g. performance, usability, reliability). These requirements concern not only the customers for whom the system is produced but ultimately every stakeholder involved with the software. Unlike functional requirements relating to the common services a system should provide, and on which stakeholders must agree, quality requirements1 usually differs from system to system, from stakeholder to stakeholder. These quality attributes can only be "satisficed" [CNYM00a], rather than "accomplished" or "satisfied", since design decisions can contribute only partially towards or against a particular quality attribute of the system. Moreover, these quality attributes are often inter-connected, whether in agreement or in conflict, with each other. Hence, reaching an agreed understanding of these qualities attributes, and finding the optimal balance among them instead of studying a single one in isolation, are crucial in achieving a high quality software product.

Software architecture sets the boundary of systematic reasoning about various quality attributes that are relevant to the system domain. A high-quality software architecture facilitates the development of a high quality software system. Designing such architecture usually involves a set of interdependent design decisions that contribute to quality attributes differently; the architect must not only iteratively explore different design alternatives for each design decision, but also consider the interplay among them and balance the myriad tradeoffs from conflicting quality attributes. This explorative process is an incremental decision making process in which the architect evaluates the design alternatives with respect to the quality attributes, and reaches an optimized design that fulfills stakeholders' requirements.

To inform these design decisions, software engineers propose architecture analysis techniques to analyze each design alternative, compare them, and understand their differences. This survey studies existing architecture analysis approaches that address all required quality attributes of the system, from the perspective of how they support an explorative design process with regard to the quality attributes. In particular, the survey explores the approaches from five perspectives: the support for gathering requirements from multiple stakeholders and resolve conflicts; the support for modeling architecture and quality attributes and identify design decisions involved in the architecture; the support for analyzing and comparing design alternatives for each design decision; the support for considering all design decisions and their interdependencies; and the automated support for the process.

[1] Quality requirements and quality attributes are interchangeable in this paper.

A Survey of Software Architecture Decision-Making Techniques

Lihua Xu, Debra J. Richardson, and Hadar Ziv
Institute for Software Research
University of California, Irvine
Irvine, CA 92697-3425
{lihuax, djr, ziv} @ics.uci.edu

ISR Technical Report # UCI-ISR-07-10

December 2007

Abstract: Software quality attributes describe both the specific criteria related to how the system is built (e.g. cost, development time), and qualitative constraints on various attributes of functions or services that the system should provide (e.g. performance, usability, reliability). These requirements concern not only the customers for whom the system is produced but ultimately every stakeholder involved with the software. Unlike functional requirements relating to the common services a system should provide, and on which stakeholders must agree, quality requirements1 usually differs from system to system, from stakeholder to stakeholder. These quality attributes can only be "satisficed" [CNYM00a], rather than "accomplished" or "satisfied", since design decisions can contribute only partially towards or against a particular quality attribute of the system. Moreover, these quality attributes are often inter-connected, whether in agreement or in conflict, with each other. Hence, reaching an agreed understanding of these qualities attributes, and finding the optimal balance among them instead of studying a single one in isolation, are crucial in achieving a high quality software product.

Software architecture sets the boundary of systematic reasoning about various quality attributes that are relevant to the system domain. A high-quality software architecture facilitates the development of a high quality software system. Designing such architecture usually involves a set of interdependent design decisions that contribute to quality attributes differently; the architect must not only iteratively explore different design alternatives for each design decision, but also consider the interplay among them and balance the myriad tradeoffs from conflicting quality attributes. This explorative process is an incremental decision making process in which the architect evaluates the design alternatives with respect to the quality attributes, and reaches an optimized design that fulfills stakeholders' requirements.

To inform these design decisions, software engineers propose architecture analysis techniques to analyze each design alternative, compare them, and understand their differences. This survey studies existing architecture analysis approaches that address all required quality attributes of the system, from the perspective of how they support an explorative design process with regard to the quality attributes. In particular, the survey explores the approaches from five perspectives: the support for gathering requirements from multiple stakeholders and resolve conflicts; the support for modeling architecture and quality attributes and identify design decisions involved in the architecture; the support for analyzing and comparing design alternatives for each design decision; the support for considering all design decisions and their interdependencies; and the automated support for the process.

[1] Quality requirements and quality attributes are interchangeable in this paper.

1 Introduction

Software quality attributes describe both the specific criteria related to how the system is built (e.g. cost, development time), and qualitative constraints on various attributes of functions or services that the system should provide (e.g. performance, usability, reliability). These requirements concern not only the customers for whom the system is produced but ultimately every stakeholder involved with the software. Unlike functional requirements relating to the common services a system should provide, the meaning of quality attributes differ from stakeholders to stakeholders -- that is, different stakeholders might have different understanding or expectation of the system with regards to specific quality attributes, e.g., the user and the security officer have different expectations over the system's security. Also unlike functional requirements on which stakeholders must agree, quality requirements usually conflict with each other -- that is, addressing one quality attribute usually leads to sacrificing others, e.g., real-time vs. reusability, flexibility vs. efficiency, reliability vs. flexibility, etc. Hence, reaching an agreed understanding of these qualities attributes and finding the right balance of quality attributes is crucial in achieving successful software products. One must identify the conflicts among desired quality attributes and work out a balance of attribute satisfaction [BE03a].

As software systems become more and more complex, addressing these quality attributes from a high-level design description has been receiving more and more attention. For instance, Parnas [Par72] introduced the concept of modularization and information hiding to improve system flexibility and comprehensibility; Perry and Wolf [PW92] further defined the notion of software architecture as involving descriptions of the elements from which systems are built, interactions among those elements, patterns that guide their composition, and constraints on these patterns. In general, a particular software architecture is defined as a collection of components that encapsulate the logic of computation, connectors that facilitate the communication among components, and their configuration. Software architecture provides high-level abstractions for representing the structure and key properties of a software system. Both the scientific and industrial communities have recognized that software architectures set the boundaries for the software qualities of the resulting system [BE03a].

To predict the quality of a software architecture design, different software architecture analysis techniques can be used to identify and verify potential risks that the quality requirements introduced to the design. There has been several attempts to understand these architecture analysis techniques that focus on how the technique assesses the potential of an architecture to deliver a system capable of fulfilling required quality attributes and to identify any potential risks [BE03b, BZJ04, DN02].

Designing a high quality software architecture is an explorative process to find an optimal combination that fulfills stakeholders' requirements. This explorative process is an incremental decision-making process in which an architect identifies several design alternatives addressing certain requirements of the system, evaluates them, balances the tradeoffs aroused with conflicting quality attributes, and reaches an optimized design that fulfills all stakeholders' requirements with minimum sacrifices. The software architecture design embodies functional design decisions and a collection of architecture design decisions that correspond to multiple quality requirements. These design decisions have a crucial influence on the success of any software project. It is necessary to have a structured way to understand the tradeoffs among different design alternatives in terms of the quality requirements, so that the developed software systems are more suitable for the problem at hand.

With the concentration on how they support the explorative design process, this survey studies existing architecture analysis approaches that evaluates architecture design alternatives with regards to multiple quality attributes to find the best fit. In particular, the survey explores the

approaches from five perspectives: the support for gathering requirements from multiple stakeholders and resolve conflicts; the support for modeling architecture and quality attributes and identify design decisions involved in the architecture; the support for analyzing and comparing design alternatives for each design decision; the support for considering all design decisions and their interdependencies; and the automated support for the process.

Although various software quality research communities have proposed their own analysis methods to ensure that specific quality attributes are addresses independently, such as real-time analysis [Kl93], reliability analysis [Ly96], and performance analysis [SW93], they are outside the scope of this survey. We argue that in real systems, quality attributes inter-connect and any design decision may involve tradeoffs among conflicting quality attributes. It is the conflicts among these quality attributes made the decision-making difficult. This survey evaluates architecture analysis approaches that support and balance systematic reasoning about all required quality attributes, instead of studying a specific quality attribute in isolation.

The remainder of this survey is organized as follows: Section 2 describes the software architecture decision-making framework. Our framework involves four critical phases, requirement determination, design elicitation, analysis of alternatives and architecture quality assurance. For each phase, we identify several interesting criteria that we want to understand and assess the studied approaches. After applying the evaluation framework and studying each approach in isolation in Section 3, we compare the evaluated results in Section 4, and draw conclusions and research recommendations in Section 5.

2 Software Architecture Analysis and Our Evaluation Framework

Architecting software is a complex design activity. It involves making decisions about a number of inter-dependent design choices that relate to a range of requirement concerns. Each decision requires selecting among a number of alternatives, each of which impacts various quality attributes in different ways. An important aspect of making decision is being able to understand the consequences of architecture design decisions with respect to the quality attributes. Software architecture analysis could provide the rationale and insight for determining which design alternative fits the overall system best with the maximized support for the quality attributes. The analysis should not only reveal requirements conflicts and incomplete design descriptions from a particular stakeholders perspective, but also establish a balance between the inter-connected quality attributes as a way to measure and achieve better quality. The quality of a system is measured by the satisfaction of a large amount of requirements originating from different stakeholders.

To understand the problem, we introduce a motivating example. Consider a hypothetical chat system, the architecture of which consists of a server and several client components (see Figure 1). In this system, the server component routes messages between the clients.

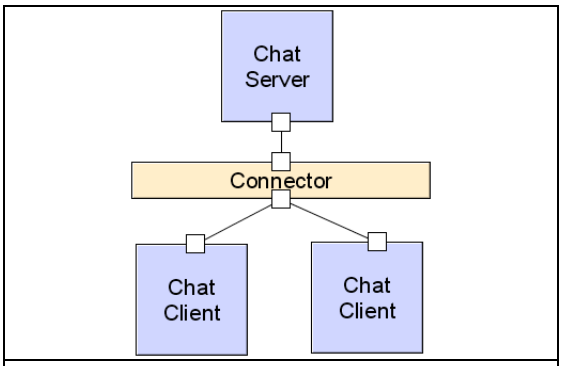

Figure 1. The Chat System Architecture

To illustrate the problem of designing a system with conflicting quality requirements, consider the following quality-attribute requirements: security, usability, and performance with their stakeholders: the security officer, and the end users.

- The security officer cares about the system's security. The security officer wants the system to be as secure as possible, not allowing unexpected users to either read or understand the messages sent around the system. Available design choices include authentication (a1), encryption (a2), or incorporating both into the system (a3); Incorporating authentication would require that clients are authenticated by the server before sending messages to other clients in the system. This helps enforce security by preventing unauthorized users from participating. Incorporating encryption into the system ensures that text messages sent by a client can only be read by intended recipients.
- The end user's interest concentrates on system's responsiveness. They expect to receive a response or deliver the status of a sent message in a bounded amount of time. Available design decisions are: bind the computation time after the user sent the message, time out after the bounded amount of time has passed, show the user a delivery failure message, and discard the message directly (b1); ask user whether to re-send or discard the message instead of discarding it directly (b2).

- The end users also care about the system's usability; they wish the system to be as easy to use as possible. Available mechanisms are to have the login window popup every time the computer starts (c1) or even to have the computer always be logged in until the user manually logs out (c2).

These quality-attribute requirements exhibit several problems:

First, the actual meaning of a quality attribute is hard to define, i.e., all three mechanisms (a1, a2, and a3) satisfy the *security* requirement to a certain extent; the notion of whether the system's security requirements have been met is thus open-ended. The stakeholders need to have a clear understanding of these unquantifiable quality attributes.

Second, the quality attributes conflict. For example, introducing encryption into the system will undoubtedly reduce its responsiveness. When conflicts happen, the stakeholders need to agree on the relative importance of these conflicting quality attributes to measure the support by the design alternative provided to the system as a whole.

Third, each design alternative makes different contributions to the related quality attributes, either conflicting or supporting. For example, as far as end user's concern, mechanism *having the login window popup every time the computer starts* (c1) provides less usability to the system than *having the computer always logged in until the user manually logs out* (c2). However, c2 decreases the system's security, thus this mechanism exhibits conflicts among *usability* and *security* requirements; On the other hand, c1 not only made the system easier to use (automatically popup the login window instead of having the user to do it manually), but also forced some level of security by only allowing certified users to be logged in. This mechanism exhibits a synergistic relationship between the *usability* and *security* requirements. It is important for the designer to understand how these different design alternatives support the related quality attributes.

Fourth, software architecture design involves many design decisions. In our illustrative example, we consider three design decisions; security, responsiveness, and usability. As mentioned, they have interdependencies and choosing one alternative would undoubtedly affect the others. The architect must decide which combination of design alternatives to choose in order to address all requirements. The art of balancing the tradeoff among design decisions from conflicting quality requirements is essential in this process.

To address these problems, different software architecture analysis techniques have been proposed to evaluate the impact of design alternatives with regards to quality attributes. To better understand these software architecture analysis techniques, we propose to view the analysis process as these following phases (as shown in Figure 2):

1. *Requirements Elicitation:* Based on multiple stakeholders' needs and insights, quality attributes are collected and determined. Stakeholders should then reach a common understanding of quality attributes so that they have an agreement on what should be expected from the system. Relative importance on the conflicting goals should also be reached so that it is agreed certain quality attributes should receive more attention when conflicts happen. Going back to our example, the required quality attributes, *security, responsiveness,* and *usability* were determined in this phase. The concrete meaning for each one should be established and agreed among the stakeholders, as well as their relative importance, so that the stakeholders understand how much they should expect the system to be secure, or responsible, or usable.
2. *Architectural Design:* Architecture design involves a set of design decisions. The designer should capture each design decision as independent unit that specifically targets a portion of the system quality requirements, so that it is easier than having to design the architecture with related quality attributes all together. The designer should also identify multiple design alternatives that satisfy the related quality attributes in different ways. As we showed in the example, having determined the required quality attributes, the designer decides to take satisfying each quality attribute as design decisions, and generates possible mechanisms for

each of them, such as introducing authentication or encryption to provide the system with certain level of security.

3. *Design Alternative Analysis*: For each design decision, every identified design alternative will be evaluated and compared in terms of their contribution or effect to related quality attributes. Because each design alternative impacts the quality attributes in different ways and the quality attributes themselves have relative importance, the designer should identify the tradeoffs among these design alternatives, measured by their contribution to the system quality. Thus, it helps to understand the consequences of each selection. Meanwhile, system requirements change from time to time. When the relative importance of quality attributes changes, there is a chance that the tradeoffs among the design alternatives change as well. The designer should acknowledge the information on whether and where any change during the decision-making process would result in the change of selection.
4. *Overall Architectural Analysis*: using the analysis result gathered from each design decision and the relationships among the different design decisions, the designer can make the design decisions that reflect the best interest of all quality attributes. Evaluating and comparing the mechanisms in isolation is not enough because of the inter-relationships among the design decisions, as discussed in our example, choosing c2 might be an obvious selection if only considering usability requirement, but it jeopardizes the security requirement; if user would like to compromise usability to security requirements, choosing c1 would be a good option, but then, the system's responsiveness is in jeopardy. The designer needs to take into account all the related design decisions and quality attributes in order to balance the conflicts and find a good architecture design.

At following sections, we will describe each phase in detail, with our evaluation framework.

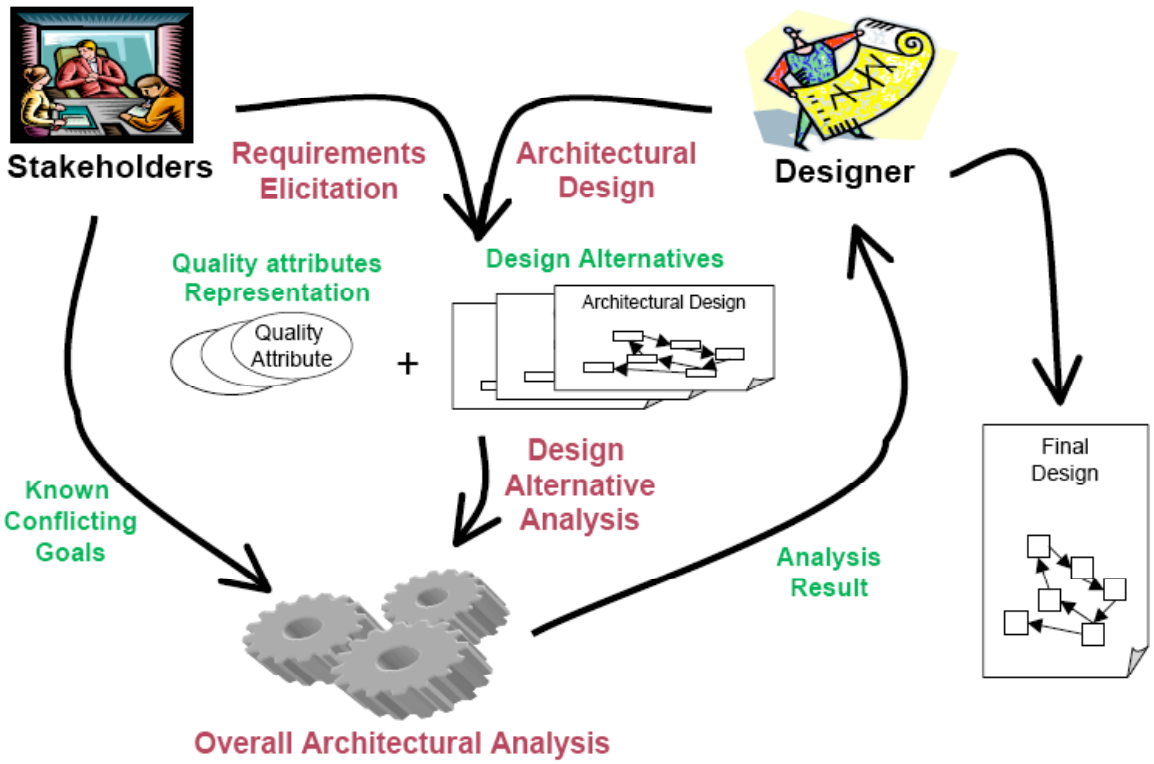

Figure 2. Software Architecture Decision Making Framework

2.1 Requirements Elicitation

During the requirements elicitation phase, the relevant quality attributes must be collected from various stakeholders and requirements consensus should be reached. Different stakeholders tend to have different views on the importance of various quality requirements for a system, partly because they experience the target system from different perspective and partly because they have conflicting goals for the target system. It is important to reach an agreement that certain quality attributes are more important than the others, so that when situations, such as a design alternative shows positive contribution to one quality attribute with negative effects on the other, happen, one can always determine how to measure the system support provided by this design alternative.

Although very important for achieving a successful software system, dealing with quality attributes are not yet very well understood. Some quality attributes lack an agreement on means for measuring the support provided by a particular architecture design alternative. We refer to these quality attributes as *unquantifiable*. For instance, the meaning for a system being "secure" or "usable" or "highly reliable" changes from system to system, from stakeholder to stakeholder. In order to evaluate an architecture design with regards to quality attributes, one needs a precise characterization of each quality attribute, that is, being able to understand an architecture design from the perspective of the quality attribute requires an understanding of how to measure or observe the quality attribute and an understanding of how various types of architecture decisions impact this measure. Therefore, during requirements determination phase, it is necessary to have support for reaching consensus among stakeholders of the unquantifiable quality attributes, so that stakeholders agree upon means for measuring the provided support for them.

2.1.1 Requirements Elicitation Evaluation Criteria

Having discussed the important activities involved, two important questions should be examined during this requirements determination phase: ***who*** provides the required quality attributes and ***how*** is consensus reached among stakeholders. The stakeholders' requirements consensus are two-folded: agreed upon means of measuring the unquantifiable quality attributes and relative importance of quality attributes resulting from conflicting goals and priorities. Hence, we present the requirements elicitation evaluation criteria as following.

As shown in Table 1, the first concern regarding this phase is who identifies the quality attributes that are used to evaluate design alternatives. There are two different approaches to the problem: (1) the quality attributes can be identified at design time, based on the design decisions; or (2) the quality attributes can be identified from stakeholders.

The second concern regarding requirements elicitation is how to understand unquantifiable quality attributes. The method could support stakeholders to establish a common understanding for specific quality attributes in the context of the target system, or provide no such support.

The third concern is how the method prioritizes the conflicting goals. The approach could provide a quantitative, qualitative method, or no support to the issue.

Requirement Elicitation	WHO	Identifying Quality Attributes	Stakeholder-based	The quality attributes under analysis are identified based on the **stakeholders' requirements**, all of which must be considered in making decisions.
			Design-based	The quality attributes under analysis are identified at **design time**.
	HOW	Understanding Unquantifiable Quality Attributes	Supported	The method provides **support** to stakeholders to establish a common understanding for analyzing the unquantifiable quality attributes in the context of the system-under-development.
			Unsupported	There is **no** such support.
		Prioritizing Conflicting Goals	Qualitative	The method uses a **qualitative** method to prioritize the conflicting goals.
			Quantitative	The method uses a **quantitative** method to prioritize the conflicting goals.
			Unsupported	The method provides **no** such support.

Table 1. Requirements determination Evaluation Criteria

2.2 Architectural Design

As discussed earlier, architecture design and evaluation are conceptually tightly related, but often performed separately in software architecture design tools. This separation causes uncertainty in architecture decision-making progress, limits the success of architecture design, and could lead to wasted effort and substantial re-work later in the development life cycle. Integrating both techniques into the decision-making process helps to appropriately consider and evaluate architecture alternatives.

Software architecture analysis could be performed at any phase during architecture development, e.g., before the architecture is fully developed, maintenance phase, architecture evolution and so forth. We also understand that different methods are optimized to achieve different evaluation goals, e.g., the analysis technique could concentrate on the design *product* (verify whether the final design meets the requirements) rather than the design *process* (support the decision-making process). It is worth noting that methods who do not perform well in our evaluation may simply because it targets on different objective. Hence, it is necessary to explicitly define and understand the development phase when the studied approaches apply so that the approach could get fair adjustment.

A precise and well-documented definition of software architecture is very important for any software architecture analysis to be performed successfully [BLF96, KBAW96, BZJ04]. An appropriate notation and abstract level to capture the architecture helps stakeholders communicate when evaluating a specific design alternative. Similar reasons apply to quality attributes; after reaching the common understanding during requirements determination phase, an appropriate representation regarding that agreement, such as concrete tasks that the system should perform to support the quality attributes, or quantified measurement on how much support has the system provide to satisfy a quality attribute, should be captured before the analysis.

Nevertheless, each architecture design involves a set of design decisions. As mentioned, making design decisions for the system as a whole is a very complex problem. The decision-making problem could become much easier if software engineers carefully divide the problem

into several sub-problems, each of which deals with one part of the system property that is easy to tackle and identify possible solutions. Therefore, the design decisions involved in the architecture design should be identified during this phase.

2.2.1 Architectural Design Evaluation Criteria

As said, quality attributes and architecture should be modeled, and the design decisions should be determined during the architectural design phase. We identify three important questions to examine while studying the approaches with respect to this phase: ***when*** does the architecture evaluation happen? ***what*** model is the evaluation based on? And ***how*** are the design decisions identified? Corresponding to the questions, Table 2 identifies the architectural design evaluation criteria.

The first concern regarding this phase is which architecture development phase does the analysis technique target. *Early Evaluation* need not wait until architecture is fully specified. The analysis and decision-making can happen at any stage during the architecture development process to examine the design alternatives. *Late Evaluation* is a form of evaluating an existing architecture. The architecture analysis and decision making take place after the architecture is fully designed.

The second concern regarding architectural design is the modeled artifacts that the evaluation based on. It could involve the architecture, or quality attributes, or both.

The third concern is how the method helps to identify the design decisions. Either the approach provide guidance on determining the design decisions involved in architecture design, or the identification is un-supported .

Architectural Design				
	WHEN	Software Architecture Development Phase	Early Evaluation	The architectural evaluation and decision making happens **during** the architecture development
			Late Evaluation	The architectural evaluation and decision making happens **after** the architecture is fully developed.
	WHAT	Modeled Artifacts	Architecture	**Architecture** is modeled.
			Quality Attributes	**Quality attributes** are modeled.
			Both	**Both** architecture and quality attributes are being modeled before the analysis.
	HOW	Identifying Design Decisions	Supported	The method provides **guidance** to identify the design decisions.
			Unsupported	The method provides **no** such guidance.

Table 2. Design Elicitation Evaluation Criteria

2.3 Design Alternative Analysis

For each design decision involved in software architecture, one needs to evaluate different alternatives with regards to multiple quality attributes. The ultimate goal of this evaluation is to choose an appropriate alternative that optimizes support to quality attributes. One common way to make this decision is to rank the design alternatives with regard to how they *satisfice* the quality attributes. During our evaluation, we will examine the source and type of design alternative comparison.

Tradeoff analysis helps designer understand the exact consequences of the chosen design alternative with respect to related quality attributes. Sensitivity analysis evaluates how the design decision could be changed due to any change during the decision making process. We will also examine how the approaches support them.

2.3.1 Design Alternative Analysis Evaluation Criteria

The design alternatives should be analyzed and compared with respect to related quality attributes, for each design decision. It is important to understand ***who*** analyzes design alternatives in terms of their support to quality attributes (the *source* of comparison) and ***how*** are the design alternatives compared. In regards to the latter question, we will evaluate the approached from three perspectives: the *type* of the comparison (either qualitative or quantitative); the support to tradeoff analysis; and the support to sensitivity analysis. Table 3 shows our design alternative analysis evaluation criteria accordingly.

The first concern regarding this phase is who analyzes the support provided by the design alternatives. *Human-based* measurement takes the stakeholders' perception as the source of comparison; and *Machine-based* measurement utilizes mathematical models or other analysis techniques to help a designer quantify the support. The second category concerns the type of comparison used by each approach when they compare and rank the design alternatives. *Qualitative* techniques compare the architecture analysis techniques qualitatively; *Quantitative* techniques quantify each design alternatives in terms of how they support the quality attributes.

The third concern is how the method supports tradeoff analysis. The approach could provide low, medium or high guidance; or tradeoff analysis is not supported in the approach. *High* guidance is provided if the approach identifies tradeoff points [KKC00], which are design decisions that influence multiple quality attributes that potentially conflict with each other, along with their relative rankings with respect to quality attributes, and relationships between design alternatives in terms of tradeoffs; *Medium* guidance is provided if the approach identifies tradeoff points along with their relative rankings; *Low* guidance is provided if the approach identifies tradeoff points.

The fourth concern is how the method deals with sensitivity. The approach could identify sensitivity points [KKC00], provide sensitivity analysis, or not support it. *Sensitivity Points* are critical properties that influence a particular quality attribute, they serve as "yellow flags" when trying to understand achievement of a quality requirement; *Sensitivity Analysis* is provided if the method helps designer reach an understanding of whether any change on the sensitivity points or other intermediate decisions during the decision-making process would effect the outcome.

Design Alternative Analysis	WHO	Design Alternative Analysis	Human-based	The method takes **stakeholders' opinion** to analyze the provided support for quality attributes
			Machine-based	The method uses **mathematical models or other analysis techniques** to quantify the support
	HOW	Design Alternatives Comparison	Qualitative	The method compares design alternatives **qualitatively**.
			Quantitative	The method compares design alternatives **quantitatively.**
		Tradeoff Analysis	High	**Tradeoff points** are identified, along with **relative ranking** with respect to quality attributes, and **relationships** between design alternatives in term of tradeoffs.
			Medium	**Tradeoff points** are identified, along with **relative ranking** with respect to quality attributes.
			Low	**Tradeoff points** are identified.
			Unsupported	There is **no** tradeoff analysis supported.
		Sensitivity	Sensitivity Analysis	The method supports **sensitivity analysis** to help designer reach an understanding of whether any change during the decision making process would effect the outcome.
			Sensitivity Points	**Sensitivity points**, which are the critical intermediate design decisions, are identified.
			Unsupported	There is **no** such support.

Table 3. Design Alternative Analysis Evaluation Criteria

2.4 Overall Architectural Analysis

Software architecture involves a collection of design decisions that respond to multiple quality attributes. Reaching an architecture design requires systematically determining the combination of the design decisions, that is, not only considering each individual design decision in isolation, selecting the design alternative that best matches stakeholders' preferences on associated quality attributes, but also need to consider the inter-dependencies among design decisions. The interplay among design decisions usually influences the selection because the chosen design alternative might have negative effects to certain quality attributes that are related to other design decisions.

Also because of the inter-relationships among design decisions, one should be able to relate the quality attributes to the related architecture elements, so that if a design decision is made, one needs to be able to identify the sacrificed quality attributes, and related to the design decision that are targeted to address them.

2.4.1 Overall Architectural Analysis Evaluation Criteria

During overall architectural analysis phase, the inter-relationships among design decisions need to be considered in order to reach a final design. The important questions to be examined are: ***how*** is the interplay among design decisions considered? And ***what*** is the output result? The criteria is shown in Table 4.

The first concern regarding this phase is whether the relationships among the stakeholders are considered during the decision-making process. The second concern is whether the method provides support to map the quality attributes with architecture elements, *No Guidance* represents

that the method claims to relate the quality attributes to architecture elements, without any specific guidance on how the mapping is achieved; *Guidance* represents that the method provides specific support of relating the quality attributes to architecture elements.

The third concern is the output result of the method. The approach could output the priority list of the design alternatives in terms of how well they support the quality attributes, or provide related information of how well each design alternative supports the quality attributes.

Overall Architectural Analysis	HOW	Context	Independent	Each design decision is considered in **isolation**.
			Dependence Relationships	The **relationships** among the design decisions are also considered when making the decision.
		Mapping	Guidance	The method provides **specific guidance** to relate the quality attributes to architectural elements.
			No Guidance	The method claims to relate quality attributes to architectural elements, ***without*** any specific guidance on how the mapping is achieved.
			Unsupported	There is **no** mapping considered in the method.
	WHAT	Quality Attributes Optimization	Decision-making information	The approach outputs the **related information** of how well the design alternatives support the quality attributes.
			Priority List	The approach outputs the **priority list** of the design alternatives in terms of how well they support the quality attributes.
			Unsupported	There is **no** such support.

Table 4. Overall Architectural Analysis Evaluation Criteria

2.5 Automation

Software architecture design and analysis is a knowledge intensive process, sometimes the design decisions made even rely on implicit assumptions and arbitrary judgments. Automating these approaches and methods, capturing and managing these technical knowledge, and rationale of the design decisions could greatly improve the architecture development process.

Some approaches adapt other techniques as part of the evaluation methodology; in order to perform the analysis method and decrease the designers' workload of switching among several tools, these adapted techniques should also be integrated into the toolset. The following section consists evaluating the architecture analysis methods with respect to their automated support.

2.5.1 Automation Evaluation Framework

Considering the automation support provided by each examined approach, it is important to understand ***what*** is the level of automation provided by the approach and ***what*** information is persistent? The criteria are shown as in Table 2.

The first concern is the level of automation. *Unsupported* represents that there is no automation available to support the method; *Low* represents that only a small portion of the method has been automated, or the only automated portion of the method is the adapted technique -- that is, the method authors' themselves did not develop any automated tool, but they adapted

others' technique, which is automated; *Medium* represents that part of the method has been automated, and the automation tool for adapted techniques is not yet integrated with the rest of the method -- that is, if one is to use the method with the automated tool, he has to switch between at least two tools, one for the adapted technique and the others for the rest of the method; *High* represents that the method has automation support as a whole toolset.

The second concern regarding automation is information persistence. The method could provide knowledge base for capturing the intermediate results during the decision making process, outputs the final results, or provide no support in this regard.

Automation	WHAT	Level of Automation	High	The method has automation support as a **whole toolset**.
			Medium	**Part** of the method has been automated, or the automation tool for adapted techniques is not yet integrated with the rest of the method.
			Low	Only a **small portion** of the method has been automated, or the only automated portion of the method is the adapted technique.
			Unsupported	There is **no** automated support.
	HOW	Information Persistence	Intermediate results	The method provides knowledge base for capturing the **reusable information** during the decision making process.
			Final results	The method only outputs the **final result**.
			Unsupported	There is **no** such support.

Table 5. Automation Evaluation Criteria

3 Survey of Software Architecture Analysis Approaches

A number of methods have been developed to help designers make better decisions during the architecture design phase. The architecture analysis approaches specifically studied in this survey are: Scenario-based Architecture Analysis Method (SAAM) [Bar02, Ka96, KBAW96], Architecture Tradeoff Analysis Method (ATAM) [Bar02, HKC00, KCW00, KKC00], Cost Benefit Analysis Method (CBAM) [KAK01], WinCBAM [BBHL94, BI96, GB01, KIC05], SAAM for Evolution and Reusability (SAAMER) [LBKK97], Scenario-based Software Architecture Reengineering (SSAR) [BB98, BB99, BLBV04], Non-functional Requirement Framework (NFR Framework) [CGY03, CNY94, CNY95a, CNY95b, CNYM00a, GY01, MCN92], Applying Analytical Hierarchy Process to Software Architecture Decisions (SAHP) [SWLM02, SWLM03], ArchDesigner [AG+05], and Tradeoff Analysis and Sensitivity Analysis for AHP related Analysis Methods (AHPTS) [ZAGJ05].

It is worth noting that the Software Engineering Institute (SEI), CMU has played a notable role in software architecture analysis research, and majority of the existing methods are related to their work. Among the studied methods, SAAM, ATAM, and CBAM are directly from their institute, WinCBAM is a cooperated work between two groups, and SAAMER adapted SAAM for analysis. Although it seems too many "SEI-related" techniques are being studied in our survey, we tried to focus on studying the ideas and evaluating the existing techniques from the perspective of how they support the explorative decision making process in architecture design. We also grouped SAAM and ATAM as one method to be studied, as they deliver the similar ideas. For the different concentrations of SAAM and ATAM, we differentiated them so that readers understand the contributions are coming from the two separate approaches.

Next, we are going to evaluate the methods using our evaluation framework described in previous sessions.

3.1 SAAM/ ATAM

Software Architecture Analysis Method (SAAM) provides a method of describing and analyzing a software architecture to show that it satisfies certain properties. The researchers concluded that various architecture descriptions do not use a common vocabulary, which makes it difficult to compare the new architectures with existing ones, hence, the method defines three perspectives for understating and describing architectures – functionality, structure, and allocation, to form a common level of understanding for comparing different architectures. The main activities involved in the SAAM are:

1. Characterize a canonical functional partitioning for the domain
2. Map the functional partitioning onto the architecture's structural decomposition
3. Choose a set of quality attributes with which to assess the architecture
4. Choose a set of concrete tasks which test the desired quality attributes
5. Evaluate the degree to which each architecture provides support for each task

ATAM is a risk identification method that provides software architects with a framework for understanding the technical tradeoffs and risks they face as they make design decisions. In an ATAM analysis, an external team facilitates meetings between stakeholders during which scenarios representing the quality attributes of the system are developed, prioritized, and analyzed against the architecture approaches chosen for the system. The results of the analysis are expressed as risks (a potentially problematic architecture decision), sensitivity points (a property of one or more components and/or relationships that is critical for achieving a particular quality attribute response), and tradeoffs (a property that affects and is a sensitivity point for more than one attribute).

3.1.1 Applying the Evaluation Framework

Method SAAM/ ATAM:

- *identifies the quality attributes based on **stakeholders'** needs.*
  Stakeholders' meetings are held to gather the set of important quality attributes with which to evaluate the architecture.
- *provides support for unquantifiable quality attributes.*
  Scenario representations are used to capture the concrete task and desired response for the target system regarding the specific unquantifiable quality attributes. Furthermore, ATAM provides a characterization framework and uses utility trees for guiding stakeholders' reach a measurable or observable point of view for the unquantifiable quality attributes.
- *provides **no** support to help reach a consensus if different stakeholders have conflicting goals.*
  When conflicting interests between different stakeholders appears, negotiation or aggregation is used to obtain a final result. However, it's the stakeholders themselves, without support from the method, which conducts the negotiation or aggregation.
- *evaluates the architecture **after** it is developed.*
  Although the quality attributes could be identified before the architecture is fully developed, the analysis itself happens until the architecture is developed, but before the implementation starts.
- ***both architecture and the quality attributes** are being modeled before the analysis.*
  SAAM defines three perspectives for describing the architectures, so that the prioritized quality attributes representation can be mapped onto the architecture representation; quality attributes are represented as scenarios.
- *provides **no** support for identifying the design decisions.*
  There is no mentioning of how the considered design decisions are identified.
- *uses **human-based** measurement to measure the provided support for quality attributes.*

The method evaluate the degree to which each architecture provides support for each task of quality attributes by taking the opinions or experienced knowledge from stakeholders.

- *uses **qualitative** method to compare the design alternatives.*
  Based on developers' experiences and previous knowledge, the designer compares the design alternatives qualitatively after applying each scenario that represents the quality attributes to the design alternatives.
- *provides **low** guidance on tradeoff analysis.*
  ATAM identifies tradeoff points but provides no guidance on the exact consequences of the chosen design alternative in terms of the tradeoffs being made for the conflicting quality attributes.
- *Identifies **sensitivity points**.*
  Sensitivity points, which are the properties of the architecture elements that are critical for achieving a particular quality attribute response, are identified during ATAM process. The result of the method on this category serves as yellow flags that remind designer or analyst to focus attention when dealing with related quality attributes.
- *provides **no** consideration of the relationships among the design decisions*
  Although the method identifies tradeoff points and sensitivity points that imply the relationship among the design decisions, it does not provide guidance or support for designers to embrace the relationship during the decision making process.
- *provides **low** guidance on relating the quality attributes to architecture elements.*
  During the step of identifying tradeoff points and sensitivity points, the mapping of quality attributes to architecture elements are performed, though the method itself does not provide specific support on how the mapping could be achieved.
- *provides **no** guidance on quality attribute optimization.*
  The method's goal is to identify the places where interested quality attributes are affected by architecture design decisions so that the designer should focus their attention on such decisions in subsequent analysis.
- ***Low** Automation.*
  SAAM is partially supported by tool SAAMTOOL.
- ***No** information persistence mechanism.*

	Requirements Elicitation		
	Identifying QA	*Understanding Unquantifiable QA*	*Prioritizing Conflicting Goals*
SAAM/ ATAM	Stakeholder-based	Supported	Unsupported

	Architectural Design		
	Development Phase	*Modeled Artifacts*	*Identifying Design Decisions*
SAAM/ ATAM	Late Evaluation	Architecture (SAAM)	Unsupported
		Quality Attributes (ATAM)	

	Design Alternative Analysis			
	Design Alternative Analysis	*Design Alternatives Comparison*	*Tradeoff Analysis*	*Sensitivity*
SAAM/ ATAM	Human-based	Qualitative	Low	Sensitivity Points

	Overall Architectural Analysis		
	Context	*Mapping*	*QA Optimization*
SAAM/ ATAM	Independent	No Guidance	Unsupported

	Automation	
	Automated Tool	*Information Persistence*
SAAM/ ATAM	Low	Unsupported

Table 6. Evaluating SAAM/ ATAM

Table 6 provides the graphical representation of applying the evaluation frameworks to method SAAM/ ATAM.

3.2 Cost Benefit Analysis Method (CBAM)

The SEI Cost Benefit Analysis Method (CBAM) is a method for architecture-based economic analysis of software-intensive systems. With the purpose of improving existing architecture design, it helps software architects to choose architecture alternatives by considering the return on investment and economic tradeoffs of these alternatives. The CBAM takes the analysis result from ATAM and associate priorities, costs and benefits with architecture decisions as additional attributes to be considered during the software architecture maintenance phase.

The CBAM consists of six steps:

- Firstly, it chooses scenarios and architecture strategies from the list output of ATAM. CBAM select a set of desired improvements to the system, possible affected portions of the existing architecture, and architecture strategies that describe the change to existing architecture design;
- Secondly, each of the stakeholders assigns a number (*quality attribute score*) to each quality attribute so that these scores total 100.
- Thirdly, it quantifies the architecture strategies' benefits. Stakeholders rank each architecture strategy in terms of its *contribution* to each quality attribute on a scale of -1 to +1, the benefit of each architecture strategy is then computed using the formula: $Benefit(AS_i) = \sum_j (Cont_{ij} \times QAscore_j)$.
- Fourthly, it quantifies the expected cost of implementing each architecture strategy that results in the expected benefit.
- Fifthly, it calculates a *desirability* by which the desired architecture strategies can be compared. By taking mean values of *benefit* and *cost*, the *desirability* is the unit *benefit/cost*.
- Finally, with all these scores, and uncertainty for each of them considered, CBAM can help architect make the strategic roadmap for software design.

The CBAM guides system engineers and other stakeholders to determine the costs and benefits associated with the architecture decisions that result in the system's qualities. Given this information, the stakeholders can then reflect upon and choose among the potential architecture decisions.

3.2.1 Applying the Analysis Framework

*Method CBAM (*Table 7*):*

- *identifies the quality attributes based on **stakeholders' needs**.*
  The method selects a set of desired improvements to the system (quality attributes), from ATAM results, whose quality attributes are gathered from stakeholder meetings.
- *provides **no** support to deal with the unquantifiable quality attributes.*
  The method chooses the intended quality attributes from ATAM, where the quality attributes have already represented as scenarios to capture the desired improvements to the system.
- *provides **quantifying** support to help reach a consensus if different stakeholders have conflicting interests over the system.*
  Each of the stakeholders gets a chance to express their interests over the system, by assigning a number as quality attribute score to each quality attribute, and each of these numbers are considered while quantifying each alternative's benefit.
- *evaluates the architecture **after** it is developed.*
  Purpose of the method is to improve the existing architecture design.
- ***No artifact** is required to be modeled before the analysis.*

Although the CBAM takes the analysis result from ATAM, where the quality attributes are represented as scenarios and architecture is fully developed, the method itself does not support to model either software architecture or quality attributes before the analysis.

- *provides **no** support for identifying the design decisions.*
  There is no mentioning of how the considered design decisions are identified.
- *uses **human-based** measurement to measure the provided support for quality attributes.*
  The method quantifies the provided support to quality attributes by taking stakeholders' opinions.
- *uses **quantitative** method to compare the design alternatives.*
  The method quantifies priorities, costs and benefits, which are associated with architecture decisions as additional attributes to be considered during the software architecture maintenance phase.
- *provides **medium** guidance on tradeoff analysis.*
  The method quantifies design alternatives' benefits to each quality attribute, which in turn, provides tradeoff information on the relative ranking of how well each quality attribute is being supported by the design alternative.
- *provides **no** guidance on sensitivity analysis.*
- *provides **no** consideration of the relationships among the design decisions*
- *provides **no** guidance on relating the quality attributes to architecture elements.*
- *provides both **make decisions** and **informs** designer on quality attribute optimization.*
  The CBAM is a decision framework. It aids designers in the elicitation and documentation of costs, benefits, and uncertainty and gives them a rational process for making choices among competing options.
- ***No** automation*

	Requirements Elicitation		
	Identifying QA	*Understanding Unquantifiable QA*	*Prioritizing Conflicting Goals*
CBAM	Stakeholder-based	Unsupported	Quantitative

	Architectural Design		
	Development Phase	*Modeled Artifacts*	*Identifying Design Decisions*
CBAM	Late Evaluation	Unsupported	Unsupported

	Design Alternative Analysis			
	Design Alternative Analysis	*Design Alternatives Comparison*	*Tradeoff Analysis*	*Sensitivity*
CBAM	Human-based	Quantitative	Medium	Unsupported

	Overall Architectural Analysis		
	Context	*Mapping*	*QA Optimization*
CBAM	Independent	Unsupported	Priority List & Decision-Making Information

	Automation	
	Automated Tool	*Information Persistence*
CBAM	Unsupported	Unsupported

Table 7. Evaluating CBAM

3.3 WinCBAM

WinCBAM [BBHL94, GB01, KIC05] is a decision-support method that integrates WinWin [ref] techniques with CBAM techniques to help stakeholders negotiate their conflict requirements by systematically evaluating software architecture alternatives as concrete conflict resolution options; hence, the stakeholders can iteratively explore, evaluate and negotiate design alternatives to reach agreement at the design stage.

The method attempts to interleave the steps of CBAM with the steps of the WinWin process in a way that mirrors and augments the natural question-and-answer process that is at the heart of requirements negotiation. During the process, stakeholders begin by entering their win conditions; if a conflict among stakeholders' win condition is identified, an issue schema is composed, summarizing the conflict and the win conditions it involves; for each issue, stakeholders explore architecture strategies as conflict-resolution options; since there are often tradeoffs among win conditions that need to be balanced, CBAM provides a means to balance these tradeoffs. CBAM is proposed here as a means to supplement the WinWin process of systematically evaluating and negotiating software architecture alternatives (as conflict-resolution options) by eliciting stakeholders' benefits and costs.

3.3.1 Applying the Analysis Framework

Method WinCBAM (Table 8)*:*

- *identifies the quality attributes based on **stakeholders**.*
  The stakeholders explicit their objectives, win conditions, and the conflicting goals, as the system's requirements.
- ***provides support** for the unquantifiable quality attributes.*
  The win conditions that the stakeholders elicited usually represents their goal and measurement of the quality attributes. Moreover, the method also provides quality attribute criteria to provide a means of understanding the quality attribute and a target by which to measure the relative merit of the proposed architecture design alternatives.
- *provides **quantitative** support to help reach a consensus if different stakeholders have contradictory opinions.*
  The method's goal is to help stakeholders' to negotiate with each other, by quantifying the architecture alternatives as conflict resolution options using CBAM.
- *evaluates the architecture **before** it is developed.*
  The method starts from requirements phase and the architecture development happens during the requirement negotiation.
- ***No artifact** is being modeled before the analysis.*
  Although the win conditions gathered from stakeholders are representations of the quality attributes, the method itself does not provide any formal format to capture them, the win conditions are just representations of what the stakeholders' objectives are.
- *provides **no** support for identifying the design decisions.*
  The considered design decisions could be identified from the architects' experiences, from previous experiences, or design patterns, etc; the method does not provide support for identifying them.
- *uses **human-based** measurement to measure the provided support for quality attributes.*
  The method quantifies the provided support according to stakeholders' opinions.
- *uses **quantitative** method to compare the design alternatives.*
  CBAM is used in the method to quantify and compare the design alternatives.
- *provides **medium** guidance on tradeoff analysis.*
  The method integrates CBAM to provide tradeoff analysis over the conflicting requirements.
- *provides **no** guidance on sensitivity analysis.*

- *provides **no** consideration of the relationships among the design decisions*
- *provides **no** guidance on relating the quality attributes to architecture elements.*
- *provides **informs** designer with the quality attributes optimization.*
  The WinCBAM is not a decision-making tool. It is a decision-support tool, helping to structure and focus the discussion of requirements by showing the participants the implications of their requirements, in terms of their realization as architecture designs.
- ***Low** Automation support.*
  The WinWin method is automated as a negotiation tool that is a Unix workstation-based groupware support system that allows stakeholders to enter win conditions, explore their interactions, and negotiate mutual agreements on the specifics of the new project being contracted. But the CBAM method is not yet automated.
- ***No** information persistence mechanism is considered in the method.*

	Requirements Elicitation		
	Identifying QA	*Understanding Unquantifiable QA*	*Prioritizing Conflicting Goals*
WinCBAM	Stakeholder-based	Supported	Quantitative

	Architectural Design		
	Development Phase	*Modeled Artifacts*	*Identifying Design Decisions*
WinCBAM	Early Evaluation	Unsupported	Unsupported

	Design Alternative Analysis			
	Design Alternative Analysis	*Design Alternatives Comparison*	*Tradeoff Analysis*	*Sensitivity*
WinCBAM	Human-based	Quantitative	Medium	Unsupported

	Overall Architectural Analysis		
	Context	*Mapping*	*QA Optimization*
WinCBAM	Independent	Unsupported	Decision-Making Information

	Automation	
	Automated Tool	*Information Persistence*
WinCBAM	Low	Unsupported

Table 8. Evaluating WinCBAM

3.4 SAAM for Evolution and Reusability (SAAMER)

SAAMER developed a modeling technique, with which SAAM applies, to ensure the rigor required for ensuring that the stakeholder objectives are explicitly addressed and traced. It consists of a framework for modeling various types of relevant information and a set of architecture views for reengineering, analyzing, and comparing software architectures.

Once the related information is gathered, it is then aligned across information categories during the modeling phase, a framework is used to model different types of information, namely, stakeholder information, architecture information, quality information, and scenarios. Both the breadth and depth of the analysis are taken into account: the breadth aspect ensures that each attribute is at least considered from the perspective of each stakeholder; the depth aspect deals with the levels of abstraction at which the stakeholder objectives are represented and analyzed.

SAAM is adopted during analysis phase and extended with sets of architecture views, e.g., static view, map view, dynamic view, and resource view, to provide different perspectives in understanding and analyzing the software systems. Explicit scenarios are mapped onto the architecture for analyze the quality attributes.

3.4.1 Applying the Analysis Framework

Method SAAMER (Table 9)*:*

- *identifies the quality attributes based on* ***stakeholders****.*
  A number of explicit scenarios are developed based on stakeholder and architecture objectives, where the scenarios are narratives that describe use cases of a system.
- *provides* ***guidance*** *for the unquantifiable quality attributes.*
  In order to better understand the system and its target quality attributes, elicitation questions are prepared for each objective and are used in interviewing domain subject experts.
- *provides* ***no*** *support to help reach a consensus if different stakeholders have contradictory opinions.*
- *evaluates the architecture* ***after*** *it is developed.*
  The method's goal is to assess an existing architecture for project evolution or reuse in a future project in the same problem domain or product line.
- ***Both architecture and the quality attributes*** *are being modeled before the analysis.*
  Architectural views are used to represent the software architecture and scenarios are used to capture the stakeholders' requirements on quality attributes.
- *provides* ***no*** *support for identifying the design decisions.*
- *uses* ***human-based*** *measurement to measure the provided support for quality attributes.*
  SAAM is adopted and SAAM uses human-based measurement.
- *uses* ***qualitative*** *method to compare the design alternatives.*
  SAAM is adopted and extended with architecture views.
- *provides* ***no*** *guidance on tradeoff analysis.*
- *provides* ***no*** *guidance on sensitivity analysis.*
- *provides* ***no*** *consideration of the relationships among the design decisions*
- *provides* ***high*** *guidance on relating the quality attributes to architecture elements.*
  Architecture map views relate the functionality with the components. Explicit scenarios are mapped onto architecture, not necessary architecture components, but rather architecture impact by the scenario, for analyzing quality attributes.
- ***informs*** *designer with the quality attributes optimization.*
  The method's result is to drive architecture development, make recommendations, locate "hot spots" in the architecture and enumerate strategies for their mitigation, identify common referent models.

- ***Low*** *Automation support.*
  Only SAAM, the adapted technique, is automated.
- ***No*** *information persistence mechanism is considered in the method.*

	Requirements Elicitation		
	Identifying QA	*Understanding Unquantifiable QA*	*Prioritizing Conflicting Goals*
SAAMER	Stakeholder-based	Supported	Unsupported

	Architectural Design		
	Development Phase	*Modeled Artifacts*	*Identifying Design Decisions*
SAAMER	Late Evaluation	Architecture & Quality Attributes	Unsupported

	Design Alternative Analysis			
	Design Alternative Analysis	*Design Alternatives Comparison*	*Tradeoff Analysis*	*Sensitivity*
SAAMER	Human-based	Qualitative	Unsupported	Unsupported

	Overall Architectural Analysis		
	Context	*Mapping*	*QA Optimization*
SAAMER	Independent	Guidance	Decision-making information

	Automation	
	Automated Tool	*Information Persistence*
SAAMER	Low	Unsupported

Table 9. Evaluating SAAMER

3.5 Scenario-based Software Architecture Reengineering (SSAR)

SSAR [BB98, BB99, BLBV04] proposes to use four different approaches to assess quality attributes in software architecture level: Scenario-based evaluation, Simulation, Mathematical Modeling, and Experience-based reasoning. For each quality attribute, the engineer can select the most suitable approach for analysis.

- *Scenario-based analysis*: A set of scenarios is developed that concretizes the actual meaning of the attribute; each individual scenario defines a context for the architecture. The performance of the architecture in that context for this quality attribute is assessed by analysis; the result from each analysis of the architecture and scenario are then summarized into an overall result, e.g., the number of accepted scenarios versus the number not accepted.
- *Simulation*: the main components of the architecture are implemented and other components are simulated resulting in an executable system.
- *Mathematical modeling:* Evaluate especially operation related software qualities with existing mathematical models or metrics from various research communities, e.g., high-performance computing, reliability, real-time systems, etc.
- *Experience-based reasoning*: Experienced software engineers provide valuable insights that may prove extremely helpful in avoiding bad design decisions and finding issues that need further analysis.

Scenario-based evaluation is the major method to be used, during which the un-satisfied quality attributes are identified and mapped onto the architecture, one at a time. In this mapping, a determination must be made by the architects as to what changes are necessary to satisfy the scenario. Changes to the architecture are performed as architecture transformations. Each transformation leads to a new version of the architecture that has the same functionality, but different values for its quality attributes.

3.5.1 Applying the Analysis Framework

Method SSAR (Table 10)*:*

- *identifies the quality attributes based on **stakeholders**.*
  The quality attributes are identified based on the requirements.
- *provides **support** for the unquantifiable quality attributes.*
  A set of scenarios is developed that concretizes the actual meaning of the quality attribute during scenario-based evaluation.
- *provides **no** support to help reach a consensus if different stakeholders have contradictory opinions.*
- *evaluates the architecture **after** it is developed.*
  The method's goal is to improve an architecture based on the quality attributes.
- ***Both architecture and the quality attributes** are being modeled before the analysis.*
  For scenario-based evaluation, the quality attributes are modeled as scenarios; for simulation, the architecture is modeled as executable.
- *provides **no** support for identifying the design decisions.*
- *uses **machine-based** measurement to evaluate the provided support to quality attributes.*
  One of the four different analysis approaches is mathematical model that utilities existing mathematical models or metrics to evaluate especially operation related software qualities.
- *uses **qualitative** and **quantitative** method to compare the design alternatives.*
  The method consists four different approaches for assessing quality attributes: scenario-based evaluation and experience-based reasoning use qualitative method, while simulation and mathematical modeling use quantitative method.
- *provides **no** support for conflict detection among required quality attributes.*

- *provides **no** guidance on tradeoff analysis.*
- *provides **no** guidance on sensitivity analysis.*
- *provides **no** consideration of the relationships among the design decisions*
  The quality attributes are dealt with one at a time.
- *provides **Low** guidance on relating the quality attributes to architecture elements.*
  After the evaluation, the un-met quality attributes are mapped on the architecture elements to identify the most prominent deficiency and transform the architecture to remove the deficiency.
- *provides **no** support to quality attributes optimization.*
- ***No** automation.*

	Requirements Elicitation		
	Identifying QA	*Understanding Unquantifiable QA*	*Prioritizing Conflicting Goals*
SSAR	Stakeholder-based	Supported	Unsupported

	Architectural Design		
	Development Phase	*Modeled Artifacts*	*Identifying Design Decisions*
SSAR	Late Evaluation	Architecture (Simulation) & Quality Attributes (Scenario-based evaluation)	Unsupported

	Design Alternative Analysis			
	Design Alternative Analysis	*Design Alternatives Comparison*	*Tradeoff Analysis*	*Sensitivity*
SSAR	Machine-based	Qualitative & Quantitative	Unsupported	Unsupported

	Overall Architectural Analysis		
	Context	*Mapping*	*QA Optimization*
SSAR	Independent	No Guidance	Unsupported

	Automation	
	Automated Tool	*Information Persistence*
SSAR	Unsupported	Unsupported

Table 10. Evaluating SSAR

3.6 Non-functional Requirement Framework (NFR Framework)

Goal satisfaction analysis between different designs in the NFR Framework [MCN92, CNY94, CNY95a, CNY95b, CNYM00, GY01] serves to systematically guide selection among architecture design alternatives. During the design process, Nonfunctional requirements are represented as goals, and their related knowledge is codified into methods and correlation rules. Methods are used to facilitate decomposition and achievement of goals, and argumentation of design decisions. Correlation rules are used to analyze the tradeoffs among design alternatives, to guide selection among alternatives, and to help detect goal conflicts.

A design alternative can positively or negatively contribute to NFRs. These contributions are modeled in the goal graph to reflect the tradeoffs made locally towards the immediate goals. Throughout the goal graph expansion process, the evaluation procedure propagates the effect of each design decision from offspring to parents. Therefore, the NFR Framework provides a way of clarifying and consolidating the tradeoffs of design alternatives considering multiple quality attributes simultaneously.

3.6.1 Applying the Evaluation Framework

Method NFR Framework:

- *identifies the quality attributes based on **stakeholders'** needs.*
  The quality attributes are gathered, decomposed, and evaluated based on stakeholders' point of view.
- ***provides support** for unquantifiable quality attributes.*
  The NFR Framework clarifies and decomposes the quality attributes either on its sort or on its parameter, such as decomposing Modifiability[System] on its parameter would result three offspring goals: Modifiability[Process], Modifiability[Data Rep], and Modifiability[Function]; and Modifiability[Function] can be further decomposed on its sort to Extensibility[Function], Updatability[Function], and Deletability[Function].
- *provides **no** support to help reach a consensus if different stakeholders have conflicting goals.*
  The method helps designer to detect and visualize the conflicting goals among different stakeholders, without any support to reach a consensus.
- *evaluates the architecture **before** it is developed.*
  The method should be used in early architecture design phase.
- ***No** artifacts are being modeled before the analysis.*
  Although the quality attributes are decomposed, the method does not specific require modeling technique to represent them.
- *provides **no** support for identifying the design decisions.*
- *uses **human-based** measurement to measure the provided support for quality attributes.*
  The method evaluate the degree to which each architecture provides support for each quality attributes by taking the opinions or experienced knowledge from stakeholders.
- *uses **qualitative** method to compare the design alternatives.*
  The method defines "satisficing" a quality attribute as satisfying the quality attribute within a limit, and the contribution to these quality attributes are defined as "strongly positive satisficing", "weak positive satisficing", "weak negative satisficing", and "strong negative satisficing". All these contribution to the quality attributes are considered when assessing the degree of goal achievement by each design alternative.
- *provides **low** guidance on tradeoff analysis.*
  When design alternatives have both "positive satisficing" and "negative satisficing" contribution to different quality attributes, the tradeoff points are identified.

- *provides **no** guidance on sensitivity analysis.*
- *provides **no** consideration of the relationships among the design decisions*
- *provides **no** guidance on relating the quality attributes to architecture elements.*
- *provides **no** guidance on quality attribute optimization.*
- ***No** Automation.*
- ***No** information persistence.*

	Requirements Elicitation		
	Identifying QA	*Understanding Unquantifiable QA*	*Prioritizing Conflicting Goals*
NFR Framework	Stakeholder-based	Supported	Unsupported

	Architectural Design		
	Development Phase	*Modeled Artifacts*	*Identifying Design Decisions*
NFR Framework	Early Evaluation	Unsupported	Unsupported

	Design Alternative Analysis			
	Design Alternative Analysis	*Design Alternatives Comparison*	*Tradeoff Analysis*	*Sensitivity*
NFR Framework	Human-based	Qualitative	Low	Unsupported

	Overall Architectural Analysis		
	Context	*Mapping*	*QA Optimization*
NFR Framework	Independent	Unsupported	Unsupported

	Automation	
	Automated Tool	*Information Persistence*
NFR Framework	Unsupported	Unsupported

Table 11. Evaluating NFR Framework

Table 11 provides the graphical representation of applying the evaluation frameworks to method NFR Framework.

3.7 Applying Analytical Hierarchy Process to Software Architecture Decisions (SAHP)

Svahnberg et al. formed a quantitative approach that supports the comparison of candidate architectures using Analytical Hierarchy Process (SAHP) [SWLM02, SWLM03]. This method provides a structured way of eliciting stakeholders' preferences for desired quality attributes and helping them gain quantified understanding of the benefits and liabilities of different architecture candidates. It produces a Framework for Quality Attribute (FQA) to capture architecture ranking according to their ability to meet particular quality attribute, and a Framework for Architecture Structures (FAS) to obtain quality attribute rankings for each architecture. Together with the prioritized quality attributes for the target system, the method is able to help designer find the best candidate design alternative for the decision. The method also provides confidence levels on the final ranking by providing a Framework for Variance Calculation (FVC).
The detailed process of applying SAHP is as following:

1. Identify potential software architecture candidates and key quality attributes.
2. Create method framework, FQA and FAS.
3. Prioritize the quality attributes for the software system to be developed
4. Identify which software architecture candidate best fits the list of prioritized quality attributes
5. Determine the uncertainty in the identification
6. Discuss the individual frameworks, the synthesized prioritized list of quality attributes and the recommended software architecture candidate to reach a consensus.

3.7.1 Applying the Analysis Framework

Method SAHP (Table 12)*:*

- *identifies the quality attributes based on **design**.*
  Although the quality attributes are gathered from stakeholders, the method actually identifies the design alternatives first and then derive the related quality attribute to evaluate.
- *provides **no** support for the unquantifiable quality attributes.*
  The method does not deal with how to reach a common understanding of the unquantifiable quality attributes under the context of the target system.
- *provides **quantifying** support to help reach a consensus if different stakeholders have conflicting interest over the system.*
  Every stakeholder is asked to give their pair-wise comparisons to prioritize the quality attributes, and the final prioritized quality attributes list is reached by taking the median value of each stakeholder's opinion.
- *evaluates the architecture both **before** and **after** it is developed.*
  The method can be used anytime during the software architecture development phase.
- ***No** artifact is being modeled before the analysis.*
  Although the architecture is developed before the analysis, the analysis itself does not require the architecture to be modeled, it could be any design decision that is irrelevant to the architecture structure.
- *provides **no** support for identifying the design decisions.*
- *uses **human-based** measurement to measure the provided support for quality attributes.*
  The method quantifies the provided support to quality attributes by taking stakeholders' opinions.
- *uses **quantitative** method to compare the design alternatives.*
  AHP method is used to create vectors signifying the relative support for different quality attributes within design alternatives (FAS), the relative ranking of how well different architecture alternatives support different quality attributes (FQA), and the prioritized quality attributes. All these vectors help the designer to compare the design alternatives.

- *provides **medium** guidance on tradeoff analysis.*
  The method uses pair-wise comparison to create FQA, which quantifies and prioritizes the relative ranking of how well each architecture alternative support different quality attributes.
- *provides **no** guidance on sensitivity analysis.*
- *provides **no** consideration of the relationships among the design decisions*
- *provides **no** guidance on relating the quality attributes to architecture elements.*
- *Outputs **priority list** for optimizing the architecture support to quality attributes.*
  Each design alternative is weighted in terms of how much they support quality attributes and how important those quality attributes are, and the design alternative that has the highest value is identified as the final design.
- ***Low** Automation.*
  Among all the techniques and steps involved in applying the method, only AHP is automated in tool Expert Choice.
- ***No** Information Persistence mechanism considered in the method.*

	Requirements Elicitation		
	Identifying QA	*Understanding Unquantifiable QA*	*Prioritizing Conflicting Goals*
SAHP	Design-based	Unsupported	Quantitative

	Architectural Design		
	Development Phase	*Modeled Artifacts*	*Identifying Design Decisions*
SAHP	Early & Late Evaluation	Unsupported	Unsupported

	Design Alternative Analysis			
	Design Alternative Analysis	*Design Alternatives Comparison*	*Tradeoff Analysis*	*Sensitivity*
SAHP	Human-based	Quantitative	Medium	Unsupported

	Overall Architectural Analysis		
	Context	*Mapping*	*QA Optimization*
SAHP	Independent	Unsupported	Priority List

	Automation	
	Automated Tool	*Information Persistence*
SAHP	Low	Unsupported

Table 12. Evaluating SAHP

3.8 ArchDesigner

In Al-Naeem et al. (2005) [AG+05], researchers have proposed ArchDesigner, a systematic quality-driven approach, for optimizing the software architecture design comprised of multiple inter-dependent design decisions. ArchDesigner improves upon previous approaches, which evaluate and select among given coarse-grained software architectures, such as CBAM, SAHP, with guidance on how to arrive at these architecture alternatives. The authors argue that since the number of candidate software architectures can be very large, it is difficult, often impossible task, to analyze all candidates, ArchDesigner is proposed to evaluate and select among candidate software architectures in a fine-grained fashion.

Design alternatives were divided into different groups, each of which represents a design decision. For a particular design decision, potential design alternatives are evaluated across a set of quality attributes associated with that design decision. AHP, with inputs as design alternatives for that design decision, their relative support for associated quality attributes, and preferences on associated quality attributes provided by different stakeholders, was applied to compute value scores for its potential alternative solutions. ArchDesigner then formulates the optimization equations so as to maximize the values associated with selected alternatives, subject to stated constraints (such as time and cost) and the inter-dependencies among design decisions.

3.8.1 Applying the Analysis Framework

Method ArchDesigner (Table 13)*:*

- *identifies the quality attributes based on **design**.*
  Although the quality attributes are gathered from stakeholders, the method actually identifies the design alternatives first and then derive the related quality attribute to evaluate.
- *provides **no** support for the unquantifiable quality attributes.*
  The method does not deal with how to reach a common understanding of the unquantifiable quality attributes under the context of the target system.
- *provides **quantifying** support to help reach a consensus if different stakeholders have conflicting interest over the system.*
  Every stakeholder is asked to give their pair-wise comparisons to prioritize the quality attributes, and the final prioritized quality attributes list is reached by taking the median value of each stakeholder's opinion.
- *evaluates the architecture **before** it is developed.*
  The fact that the method breaks down the architecture design into a set of design decisions to make and considers the inter-dependence among these design decisions when consolidating the decision made the method more suitable for assisting designer while the architecture is under development.
- ***No** artifact is being modeled before the analysis.*
  The analysis does not require the architecture to be modeled; it could be any design decision that is irrelevant to the architecture structure.
- *provides **guidance** for identifying the design decisions.*
  Since the number of architecture candidates could be large, in order to help stakeholders arriving at a suitable software architecture solution, the method breaks down the architecture design into a set of design decisions and optimizes the software architecture design comprised of these inter-dependent design decisions.
- *uses **human-based** measurement to measure the provided support for quality attributes.*
  The method quantifies the provided support to quality attributes by taking stakeholders' opinions.
- *uses **quantitative** method to compare the design alternatives.*

The method uses AHP to compute value score of each design alternative as the degree to which an alternative satisfies the desired quality attributes.

- *provides **medium** guidance on tradeoff analysis.*
  The method quantifies and prioritizes the relative ranking of how well each architecture alternative support different quality attributes.
- *provides **no** guidance on sensitivity analysis.*
- *considers **inter-dependent** relationships among the design decisions*
  Optimization techniques, particularly Integer Programming, are used to make sure that the selection of any design alternative should maintain the dependencies and obey the global constraints of the system.
- *provides **no** guidance on relating the quality attributes to architecture elements.*
- *Outputs **priority list** for optimizing the architecture support to quality attributes.*
  Each design alternative is weighted in terms of how much they support quality attributes and how important those quality attributes are, and the design alternative, which has the highest value without violating the inter-dependence relationships and global constraints, is identified as the final design.
- ***Low** Automation.*
  Among all the techniques and steps involved in applying the method, only AHP is automated in tool Expert Choice.
- ***No** Information Persistence mechanism considered in the method.*

	Requirements Elicitation		
	Identifying QA	*Understanding Unquantifiable QA*	*Prioritizing Conflicting Goals*
ArchDesigner	Design-based	Unsupported	Quantitative

	Architectural Design		
	Development Phase	*Modeled Artifacts*	*Identifying Design Decisions*
ArchDesigner	Early Evaluation	Unsupported	Guidance

	Design Alternative Analysis			
	Design Alternative Analysis	*Design Alternatives Comparison*	*Tradeoff Analysis*	*Sensitivity*
ArchDesigner	Human-based	Quantitative	Medium	Unsupported

	Overall Architectural Analysis		
	Context	*Mapping*	*QA Optimization*
ArchDesigner	Independent	Unsupported	Priority List

	Automation	
	Automated Tool	*Information Persistence*
ArchDesigner	Low	Unsupported

Table 13. Evaluating ArchDesigner

3.9 AHP with Tradeoff and Sensitivity Analysis (AHPTS)

Zhu et al. provides crucial additional in-depth tradeoff and sensitivity analysis on top of a standard AHP (AHPTS) [ZAGJ05]; In addition to a ranking, the method provides the designer with tradeoff analysis results that shows the exact consequences of the chosen design alternatives in terms of the key tradeoffs being made and the extent of these tradeoffs when compared to tradeoffs implied by other alternatives, and sensitivity analysis information that shows whether slightly different intermediate decisions on previous pair-comparisons or priority weights could change the outcome.

AHPTS provides mechanisms to analyze tradeoffs for design alternatives both with and without quality attribute weights. It utilizes the two-dimensional sensitivity diagram in Expert Choice where any two chosen quality attributes represent the *x* and *y* axis and each design alternative's contributions to these two quality attributes are plotted. The area in the diagram is then divided into four quadrants that in turn divide the relative size of the tradeoff into four groups. Therefore, design alternatives falling into the upper left and bottom right quadrant indicate the relatively important tradeoffs being made if the design alternative is chose. The size of the tradeoff is indicated by the extent a point is positioned towards the upper left or bottom right corner. Design alternatives falling into the bottom left quadrant indicate both quality attributes are negatively affected. Design alternatives falling into the upper right indicates both quality attributes are positively affected. The results for a selected design alternative can be documented as part of the design rationales and consequences for future analysis and evolution in subsequent development phases, and it could help stakeholders to focus on the project-wide important tradeoffs.

For sensitivity analysis, AHPTS utilizes gradient diagram in Expert Choice, each of which is designed for each quality attribute. The *vertical* line represents the priority weight of the quality attribute and is read from *x* axis; the priorities for the design alternatives are read from *y* axis. They are determined by the intersection of the alternative's line with the quality attribute (vertical) priority line. As the vertical line moves along the *x* axis by changing its priority weight, the intersections with the horizontal lines represents the new priority of the design alternative read from *y* axis. When the vertical line meets an intersection of two design alternatives, the final ranking of the two alternatives will be altered. So if the value of the distance between the current vertical line and the intersection of two design alternatives is the smallest number, then the weight for this quality attribute or the relative weight of design alternatives is the most sensitive and critical decisions. Hence, if architects and stakeholders are not interested in the most sensitive points, they can examine the diagrams visually to inspect any intersections which are relatively close to the current priority. If two lines denoting design decisions never cross, this means that no matter how the quality attributes are weighted, the ranking of the design alternative will never change.

3.9.1 Applying the Evaluation Framework

Method AHPTS (Table 14)*:*

- *identifies the quality attributes based on **design**.*
  Although the quality attributes are gathered from stakeholders, AHPTS identifies the design alternatives first and then derive the related quality attribute to evaluate.
- *provides **no** support for the unquantifiable quality attributes.*
  The method does not deal with how to reach a common understanding of the unquantifiable quality attributes under the context of the target system.
- *provides **quantifying** support to help reach a consensus if different stakeholders have conflicting interest over the system.*

The method built upon AHP related techniques, where every stakeholder is asked to give their pair-wise comparisons to prioritize the quality attributes, and the final prioritized quality attributes list is reached by taking the median value of each stakeholder's opinion.

- *evaluates the architecture both **before** and **after** it is developed.*
  The method concentrates on tradeoff analysis and sensitivity analysis, which made the method suitable for using during or after the architecture design.
- ***No** artifact is being modeled before the analysis.*
  The analysis does not require the architecture to be modeled; it could be any design decision that is irrelevant to the architecture structure.
- *provides **no** guidance for identifying the design decisions.*
- *uses **human-based** measurement to measure the provided support for quality attributes.*
  The method quantifies the provided support to quality attributes by taking stakeholders' opinions.
- *uses **quantitative** method to compare the design alternatives.*
  The method uses AHP to compute value score of each design alternative as the degree to which an alternative satisfies the desired quality attributes.
- *provides **high** guidance on tradeoff analysis.*
  The method quantifies and prioritizes the relative ranking of how well each architecture alternative support different quality attributes, provides visualization of the tradeoffs and their relative sizes among the design alternatives.
- *provides **sensitivity analysis**.*
  One of the goals of the work is to deal with changing quality priority. The most sensitive critical decisions are obtained by looking for the smallest change that will alter a final alternative ranking. The change value can be an indicator of architecture sensitivity.
- *considers **no** relationships among the design decisions*
  AHPTS does not focus on the inter-dependent relationships among the design decisions.
- *provides **no** guidance on relating the quality attributes to architecture elements.*
- *Outputs **priority list** and **decision-making information** for optimizing the architecture support to quality attributes.*
  Other than normal AHP result, which is the relative ranking of the design alternatives, the method enriches it by making design consequences explicit and indicates the architecture sensitivity of whether changing a quality attribute's priority will alter the final ranking of the design alternatives.
- ***Medium** Automation.*
  The tradeoff and sensitivity analysis techniques are integrated with Expert Choice, which is the automated tool for AHP techniques. The analysis technique is not yet integrated with design techniques that should also be involved in the decision making process.
- ***No** Information Persistence mechanism considered in the method.*

	Requirements Elicitation		
	Identifying QA	*Understanding Unquantifiable QA*	*Prioritizing Conflicting Goals*
AHPTS	Design-based	Unsupported	Quantitative

	Architectural Design		
	Development Phase	*Modeled Artifacts*	*Identifying Design Decisions*
AHPTS	Early & Late Evaluation	Unsupported	Unsupported

	Design Alternative Analysis			
	Design Alternative Analysis	*Design Alternatives Comparison*	*Tradeoff Analysis*	*Sensitivity*
AHPTS	Human-based	Quantitative	High	Sensitivity Analysis

	Overall Architectural Analysis		
	Context	*Mapping*	*QA Optimization*
AHPTS	Independent	Unsupported	Priority List & Decision-Making Information

	Automation	
	Automated Tool	*Information Persistence*
AHPTS	Medium	Unsupported

Table 14. Evaluating AHPTS

4 Comparing Surveyed Approaches

After evaluating each approach in isolation, we combined the evaluation results together as shown in Table 15, and compared those approaches with respect to different properties of the evaluation framework. Tables 16-21 and their descriptions will illustrate our interesting observations:

Table 15 shows the evaluation results of all the studied approaches, to clearly visualize the results, we use "-" to represent *Unsupported* in each category:

	Requirements Elicitation		
	Identifying QA	*Unquantifiable QA*	*Conflicting Goals*
SAAM / ATAM	Stakeholder-based	Supported	-
CBAM	Stakeholder-based	-	Quantitative
WinCBAM	Stakeholder-based	Supported	Quantitative
SAAMER	Stakeholder-based	Supported	-
SSAR	Stakeholder-based	Supported	-
NFR Framework	Stakeholder-based	Supported	-
SAHP	Design-based	-	Quantitative
ArchDesigner	Design-based	-	Quantitative
AHPTS	Design-based	-	Quantitative

	Architectural Design		
	Development Phase	*Modeled Artifacts*	*Identifying Design Decisions*
SAAM /ATAM	Late Evaluation	Architecture & Quality Attributes	-
CBAM	Late Evaluation	-	-
WinCBAM	Early Evaluation	-	-
SAAMER	Late Evaluation	Architecture & Quality Attributes	-
SSAR	Late Evaluation	Architecture & Quality Attributes	-
NFR Framework	Early Evaluation	-	-
SAHP	Early & Late Evaluation	-	-
ArchDesigner	Early Evaluation	-	Guidance
AHPTS	Early & Late Evaluation	-	-

	Analysis of Alternatives			
	Measure Alternatives	*Compare Alternatives*	*Tradeoff Analysis*	*Sensitivity*
SAAM/ ATAM	Human-based	Qualitative	Low	Sensitivity Points
CBAM	Human-based	Quantitative	Medium	-
WinCBAM	Human-based	Quantitative	Medium	-
SAAMER	Human-based	Qualitative	-	-
SSAR	Machine-based	Qualitative & Quantitative	-	-
NFR Framework	Human-based	Qualitative	Low	-
SAHP	Human-based	Quantitative	Medium	-
ArchDesigner	Human-based	Quantitative	Medium	-
AHPTS	Human-based	Quantitative	High	Sensitivity Analysis

	Architecture Quality Assurance		
	Context	*Mapping*	*QA Optimization*
SAAM / ATAM	Independent	No Guidance	-
CBAM	Independent	-	Priority List & Decision-making Information
WinCBAM	Independent	-	Decision-making Information
SAAMER	Independent	Guidance	Decision-making Information
SSAR	Independent	No Guidance	-
NFR Framework	Independent	-	-
SAHP	Independent	-	Priority List
ArchDesigner	Dependence Relationships	-	Priority List
AHPTS	Independent	-	Priority List & Decision-making Information

	Automation	
	Automated Tool	*Information Persistence*
SAAM / ATAM	Low	-
CBAM	-	-
WinCBAM	Low	-
SAAMER	Low	-
SSAR	-	-
NFR Framework	-	-
SAHP	Low	-
ArchDesigner	Low	-
AHPTS	Medium	-

Table 15. Evaluation Results

Next, we'll examine and compare the evaluation results with respect to related categories across criteria.

Table 16 examines how the studied approaches support for reaching consensus on the requirements. As discussed, quality attributes, especially unquantifiable ones, are usually hard to measure. To precisely evaluate the design alternatives with respect to quality attributes, stakeholders should reach an agreement on understanding the concrete tasks or meanings expected from the system qualities early in the analysis process. One particular design alternative usually impact more than one quality attributes, in conflict or in synergistic ways. It is necessary for stakeholders to also agree on the relative importance among the quality attributes so that the designer knows what to concentrate when conflict exists. Therefore, we compared each approach's evaluation results with regard to two properties, *Understanding Unquantifiable Quality Attributes* and *Prioritizing Conflicting Goals*, to examine how they support for reaching consensus on the requirements.

	Requirements Elicitation	
	Understanding Unquantifiable QA	*Prioritizing Conflicting Goals*
SAAM/ ATAM	Support	-
CBAM	-	Quantitative
WinCBAM	Support	Quantitative
SAAMER	Support	-
SSAR	Support	-
NFR Framework	Support	-
SAHP	-	Quantitative
ArchDesigner	-	Quantitative
AHPTS	-	Quantitative

Table 16. Reaching Consensus on Requirements

Limited support for reaching consensus on the requirements: As shown in Table 16, Only WinCBAM provides support to both *unquantifiable quality attributes* and *conflicting goals*. WinCBAM provides requirements negotiation to identify issues/ conflicts among stakeholders, generates design options to resolve these issues, evaluate each option with regard to the quality attributes, and finally reaches an agreement. As mentioned earlier, the designer should get information on the required quality attributes, upon which the design alternatives could b e compared, during requirements determination phase. The unquantifiable nature of quality attributes made their actual meanings differ from system to system, from stakeholder to stakeholder. Without an established understanding of the actual meaning of these quality attributes, it is impossible to compare the design alternatives meaningfully. Also, different stakeholders have different goals over the system, and satisfying one usually has to sacrifice the others. It is also impossible to meaningfully compare the design alternatives if conflicting goals are involved. Therefore, the analysis technique should provide support for reaching consensus on both meaning of unquantifiable quality attributes and relative importance of conflicting goals.

Table 17 examines how the studied approaches utilize the quality attributes identified in Requirements Eliciation phase. Ideally, the quality attributes that guide the architecture analysis should be identified from stakeholders, representing their expectations of system services or functions. To ensure that all of stakeholders' requirements are considered, represented and met in the final software architecture design, the set of identified quality attributes must all be utilized; that is, in order to help stakeholders arriving a suitable software architecture solution that addresses all the required quality attributes, the designer needs to break down the architecture design into a set of design decisions with respect to these quality attributes, so the architecture design that is comprised of these inter-connected design decisions can be optimized with regard to stakeholders' requirements. Therefore, we compare each approach's evaluation results with regard to two properties, *Identifying Quality Attributes* and *Identifying Design Decisions*, to examine how they utilize the identified quality attributes.

	Requirements Elicitation	**Architectural Design**
	Identifying QA	*Identifying Design Decisions*
SAAM/ ATAM	Stakeholder-based	-
CBAM	Stakeholder-based	-
WinCBAM	Stakeholder-based	-
SAAMER	Stakeholder-based	-
SSAR	Stakeholder-based	-
NFR Framework	Stakeholder-based	-
SAHP	Design-based	-
ArchDesigner	Design-based	Guidance
AHPTS	Design-based	-

Table 17. Identified Quality Attributes

No support to make sure that all of the identified quality attributes are considered during the process (Table 17): During the study, we noticed that although some approaches (SAAM/ATAM, CBAM, WinCBAM, SAAMER, SSAR, NFR Framework) claim that they identify the quality attributes from stakeholders, it is unclear whether the whole set of identified quality attributes are actually being considered, or the approach utilizes part of the identified quality attributes that are influenced by the design decisions that identified without structured guidance. These approaches provide no support on determining the design decisions under consideration, and they provide no other mechanism to ensure that all the required quality attributes are being considered in the decision-making process. On the other hand, ArchDesigner breaks down the architecture design into a set of design decisions. But the method provides no mechanism to collect quality attributes from stakeholders. The quality attributes that considered by the method are the ones that influenced by the design decisions, there is no ensure whether all stakeholders' requirements are met.

Table 18 examines how the studied approaches support for the inter-relationships among the design decisions involved in the architecture design. As discussed, in order to arrive a suitable software architecture solution that addresses all the required quality attributes, the designer needs to break down the architecture design into a set of design decisions with respect to these quality attributes. These design decisions inter-connect with each other because multiple design decisions could influence the same quality attributes in different ways, and this could alter the decision. Having the mapping relationships among quality attributes and architectural elements clearly identified helps designer make a global selection easier, especially when the chosen design alternative have negative affects to certain quality attributes that are related to other design decisions.

	Architecture Quality Assurance	
	Context	*Mapping*
SAAM/ ATAM	Independent	No Guidance
CBAM	Independent	-
WinCBAM	Independent	-
SAAMER	Independent	Guidance
SSAR	Independent	No Guidance
NFR Framework	Independent	-
SAHP	Independent	-
ArchDesigner	Dependence Relationship	-
AHPTS	Independent	-

Table 18. Multiple Design Decisions

Limited support for considering the multiple design decisions involved in the architecture design: ArchDesigner provides support to consider the inter-dependence relationship among design decisions, however, there is no mapping to relate the influenced quality attributes with architecture elements, and ultimately other design decisions that also have influences to the same quality attributes. As mentioned, reaching an architecture design requires to systematically determining the combination of the design decisions. An ideal architecture decision-making technique should consider and support both the inter-dependence relationships among design decisions and the mapping relationships among quality attributes and architectural elements.

Table 19 examines the evaluation results regarding modeled artifacts criterion.

	Design Elicitation	
	Modeled Artifacts	
SAAM/ ATAM	SA	QA
CBAM	-	-
WinCBAM	-	-
SAAMER	SA	QA
SSAR	SA	QA
NFR Framework	-	-
SAHP	-	-
ArchDesigner	-	-
AHPTS	-	-

Table 19. Modeled Artifacts

Limited support for modeling both the architecture and the quality attributes: Two approaches, SAAMER and SAAR, provide explicit modeling mechanism for represent both the architecture and quality attributes. Although method SAAM/ ATAM showed as provide modeling both artifacts, SAAM is the method that actually provides modeling mechanism for architectures and ATAM is the method models quality attributes. To make a design decision in software architecture that fulfills stakeholders' requirements, an agreed representation of the architecture should be reached so the stakeholders have a common understanding of the abstraction level of the architectural elements that support the required quality attributes. Same reason applies to quality attributes; the quality attributes should be represented so the stakeholders' have an agreement on the actual meaning and expectation of them. Therefore, an ideal architecture decision-making technique should model both the architecture and quality attributes.

Table 20 examines the evaluation results in Analysis of Alternatives criteria.

	Analysis of Alternatives			
	Measure Alternatives	*Compare Alternatives*	*Tradeoff Analysis*	*Sensitivity*
SAAM/ ATAM	Human-based	Qualitative	Low	Sensitivity Points
CBAM	Human-based	Quantitative	Medium	-
WinCBAM	Human-based	Quantitative	Medium	-
SAAMER	Human-based	Qualitative	-	-
SSAR	Machine-based	Qualitative & Quantitative	-	-
NFR Framework	Human-based	Qualitative	Low	-
SAHP	Human-based	Quantitative	Medium	-
ArchDesigner	Human-based	Quantitative	Medium	-
AHPTS	Human-based	Quantitative	High	Sensitivity Analysis

Table 20. Analysis of Alternatives

Limited support for measuring the provided support of quality attributes: Only one approach, SSAR, allows designer to use mathematical models or metrics from various quality attribute community to evaluate the support provided by the design alternative; other approaches ask stakeholders' opinion or uses experienced knowledge to measure the provided support of the quality attributes. Human's opinions are inconsistent, and prone to faults. Structured reasoning or analysis techniques could help eliminate the incorrect measuring for design alternatives.

Qualitative measurement vs. Quantitative measurement: Different approaches choose to use different measurement when comes to compare the unquantifiable quality attributes. It is difficult to say whether qualitative measurement is better than quantitative measurement, or vice versa, because of different nature of different quality attributes. Only one method, SSAR, provides both measurements to deal with these different quality attributes.

Limited support for tradeoff analysis: Although most approaches provide some level of tradeoff analysis, there is still limited information provided by the tradeoff analysis. For instance, method ATAM identifies tradeoff points as architecture elements affecting different quality attributes simultaneously, however, when a tradeoff decision needs to be made, ATAM leaves the decision process largely to requirement negotiation; CBAM, which all potential design alternatives are linked to their benefits through a response-utility function and value analysis is performed to determine the best candidate, and AHP related multi-criteria decision making techniques, such as SAHP, ArchDesigner, and AHPTS, which derive weighted priorities from pair-wise qualitative comparisons, provides more formal quantitative methods than informal negotiation. However, only AHPTS not only quantifies and prioritizes the relative ranking of how well each architecture

alternative support different quality attributes, the visualization of the tradeoffs and their relative sizes among the design alternatives are also provided, so that the designer could choose design alternatives based on the related information provided by tradeoff analysis.

Limited support for sensitivity analysis: There is even less support considered for sensitivity analysis, method ATAM identifies sensitivity points and leaves sensitivity analysis to the designer; only one method, AHPTS, provides high level sensitivity analysis during the decision-making process. While making the architecture design decisions, it is possible that the intermediate data could change over time. For instance, usability was considered to be the most important quality attribute in the system, so the chosen design alternative are more likely to provide more usability to the system while sacrificing the other quality attributes; it is possible that stakeholders realized later that security should be paid more attention than usability, then the chosen design alternative might not be the best fit for the system any more. Sensitivity analysis could provide design with such information that if the relative importance among quality attributes altered, whether the related design decision should be changed accordingly.

Table 21 examines the evaluation results for automated support provided by each approach.

	Automation	
	Automated Tool	*Information Persistence*
SAAM/ ATAM	Low	-
CBAM	-	-
WinCBAM	Low	-
SAAMER	Low	-
SSAR	-	-
NFR Framework	-	-
SAHP	Low	-
ArchDesigner	Low	-
AHPTS	Medium	-

Table 21. Automation Support

Limited tool support: There is only two techniques being automated; SAAM techniques has an automated tool, SAAMTOOL, which partially support the evaluation process, and Expert Choice is a commercial tool that automated the AHP process for decision makers. Ideally, the approach should automate and integrate related decision-making activities into a toolset, such as design activities, analysis techniques, etc, so that the architecture explorative design process is supported.

No Information Persistence Mechanism: Surprisingly, there is no approach provide mechanism to manage the historic data during the decision making process. A number of tedious tasks, as collecting, documenting, managing the historic information, etc, exist during the architecture decision-making process. They serve as design rationales that are invaluable to future designs. Tools that capture the design artifacts should also provide features to capture this information.

5 Conclusions and Research Recommendations

A high quality software architecture substantially increases the likelihood of building a high quality software system. In working toward a higher-quality architecture, the software designer must take into account not only the functional capabilities required of the system, but also the quality requirements from various stakeholders; One needs to identify design alternatives for each design decision; evaluate them and find a best fit for the system. This survey studied existing architecture analysis techniques from the perspective of their support to the architecture decision-making process. We choose these representative approaches to study because they evaluate architecture alternatives with respect to all required quality attributes to select a best-fit architecture, instead of focusing on a single quality attribute.

As discussed in Section 4, the survey shows that existing approaches still lack in several important ways and could benefit from improvements in those key areas. Following is our list of research recommendations that could contribute to these improvements.

- **Appropriate representation of both quality attributes and architecture**.

The quality attributes, as stated, are too vague to be directly evaluated, e.g. the actual meaning or measurement of quality attribute "flexibility" varies among systems that involve different stakeholders. Each architecture design could embody some decisions that restrict flexibility. Thus, the system can hardly say "flexible", unless it meant appropriately flexible with respect to an anticipated set of uses within, within a limit. As we observed, most of studied approaches use scenarios to represent the expected system quality attributes, these scenarios serve as both illustrations of the specific aspect or particular instantiations of the quality attributes, and contracts among stakeholders that they agreed on the limit that the quality attributes should be satisfied.

The architecture analysis techniques are intended to support the architecture decision-making process and verify that the designed architecture is actually able to support the required quality attributes, hence, a valid representation of architecture could help stakeholders have a clear description of it that exposes its main features and the quality attributes. However, as we observed, only two approaches, SAAMER and SAAR, provide representations for both architecture and quality attributes. SAAMER uses a set of architecture views to provide different perspectives in understanding and analyzing the software system architectures; and SSAR provides four different analysis methods, among which scenario-based analysis uses scenarios to represent the quality attributes and simulation represents architectures as executables.

Therefore, we recommend that appropriate representation of both quality attributes and architecture should be maintained before the architecture analysis, to build a solid understanding and background for various stakeholders.

- **Mechanism to ensure that all of the identified quality attributes are being considered.**

Quality attributes usually interfere with each other, either conflicting or supporting. These quality attributes are usually gathered from various stakeholders, representing different interests of various parties. It is not appropriate to concentrate on some quality attributes while ignoring others without careful investigation. Thus, the software architecture, on which the software development is based, should not only provide the required functions and services of the system, but also consider the required quality attributes, balance their conflicts, and find a best fit architecture that satisfies the required quality attributes. Hence, mechanism to ensure that all of the identified quality attributes are being considered during the architecture decision-making process, such as using quality attributes to guide the identification of design decisions, is

necessary to make sure the design architecture indeed meets the whole set of requirements that are gathered from various stakeholders.

An architecture design involves multiple design decisions that target on different quality attributes or functions. As Al-Naeem et al. [AG+05] suggested, a good architecture design alternative for a design decision should not only meet its local requirements (the requirements for optimizing the design decision by itself), but also its global requirements (the constraints in the system). Utilizing the required quality attributes to guide the identification of design decisions, and optimizing them with regards to constraints from inter-connected design decisions could help ensure that all required quality attributes are considered. This optimization requires mapping relationships among quality attributes and architectural elements, so that the influenced artifacts could be traced across design decisions.

- **Support for analyzing design alternatives and detecting conflict quality attributes**.

As we observed, most of the approaches relies on stakeholders' opinions to analyze the provided support by design alternatives, and it also relies on stakeholders to find conflicts among their goals. However, human opinions are prone to faults; even worse, the stakeholders themselves sometimes do not even know how much support a design alternative could give to the quality attributes, and do not know the conflicts exist until they see how the design alternatives work. Therefore, the architecture analysis technique should also provide support to measure the provided support by design alternatives, and detect conflicts among quality attributes.

- **Support for both qualitative and quantitative analysis**.

As mentioned in Section 4, both qualitative and quantitative analysis have their benefits and weaknesses when analyzing different quality attributes. An ideal approach should incorporate both analysis techniques to provide accurate evaluation.

- **Support for tradeoff analysis and sensitivity analysis**.

In order to make an appropriate selection of the design alternatives, and fully understand the consequences of the selection, the designer needs to be informed with the tradeoffs among design alternatives, as well as the sensitivity points and indications of the architecture sensitivity whether change in sensitivity points will alter the final ranking of the design alternatives.

- **Integrated automation support**.

As discussed earlier, architecture design and evaluation are conceptually tightly related, but often performed separately in software architecture design tools. This separation causes uncertainty in architecture decision-making progress, limits the success of architecture design, and could lead to wasted effort and substantial re-work later in the development life cycle. Integrating both techniques into the decision-making process is needed to appropriately consider and evaluate architecture alternatives.

Most of the studied approaches do not provide enough automation support for the analysis techniques, not to mention that none of the approach integrates the analysis technique with the design environment to support the architecture decision-making process. In order to increase the confidence of the analysis result, the analysis techniques should be automated. Designing a software architecture requires exploring and managing a lot of design alternatives, revisiting and utilizing evaluation data, and finally reaching a best fit design to satisfy the involved quality attributes. During this process, a lot of tedious tasks are involved, automating the process could save the designer much energy and time. Traditionally, there exist a lot of design environment that help to design the system. We suggest that the automation support should integrate the analysis technique with the design environment so that the architecture explorative design process could be well supported, and mechanism should be provided to help designer to manage the historic data during the process.

6 References

[AB03] AY. Ababutain, AGR. Bullen, Multicriteria decision-making model for selection of build-operate-transfer tall road proposals in the public sector, Transportation Research Record (1848): 1-9, 2003.

[ABE02] O. Aldawud, A. Bader and T. Elrad, Weaving with Statecharts, Aspect-Oriented Modeling with UML workshop at the 1st International Conference on Aspect-Oriented Software Development, April 2002.

[AG+05] T. Al-Naeem, I. Gorton, M.A. Babar, F. Rabhi, and B. Benatallah, "A quality-driven systematic approach for architecting distributed software applications", Proceedings of the 27th International Conference on Software Engineering (ICSE), St. Louis, USA, 2005.

[Ale77] C. Alexander, A Pattern Language: Towns, Buildings, Construction, Oxford University Press, 1977.

[Ale79] C. Alexander, The Timeless Way of Building, Oxford University Press, 1979.

[Bar02] M. Barbacci, "SEI Architecture Analysis Techniques and When to Use Them", technical report, CMU/SEI-2002-TN-005.

[BB98] P. Bengtsson, J. Bosch, "Scenario Based Software Architecture Reengineering", 5th International Conference of Software Reuse (ICSR), 1998.

[BB99] P. Bengtsson, J. Bosch, "Architecture Level Prediction of Software Maintenance," Proceedings of 3rd European Conference on Software Engineering Maintenance and Reengineering, 1999.

[BBHL94] B. Boehm, P. Bose, E. Horowitz, M. J. Lee, "Software Requirements as Negotiated Win Conditions", Proceedings Intl. Conference on Requirements Engineering, IEEE April 1994.

[BCK03] L. Bass, P. Clements, and R. Kazman, "Software Architecture in Practice", Addison-Wesley, 2003.

[BCK98] L. Bass, P. Clements, and R. Kazman, Software Architecture in Practice. Addison-Wesley, 1998.

[BE03a] M.R. Barbacci, R. Ellison, et al, Quality-attribute Attribute Workshops (QAW), Technical Report, CMU/SEI-2003-TR-016.

[BE03b] R. Bahsoon and W. Emmerich, "Evaluating software architectures: development, stability and evolution", ACS/IEEE International Conference on Computer Systems and Applications, Tunis, Tunisia, July 14-18, 2003.

[BI96] Barry Boehm, and Hoh In, Identifying Quality-attribute-Requirement Conflict. IEEE Software, 1996.

[BLBV04] P. Bengtsson, N. Lassing, J. Bosch, and H.V. Vliet, "Architecture-Level Modifiability Analysis", Journal of Systems and Software, vol. 69, 2004.

[BLF96] S. Bot, C.H. Lung, and M. Farrell, "A Stakeholder-Centric Software Architecture Analysis Approach", Proceedings of the 1st International Software Architecture Workshop (ISAW), 1996, pp. 152-154.

[BZJ04] M.A. Babar, L. Zhu, R. Jeffery, "A Framework for Classifying and Comparing Software Architecture Analysis Methods", Proceedings of the 2004 Australian Software Engineering Conference (ASWEC'2004), 2004.

[CBB+03] P. Clements, F. Bachmann, L. Bass, D. Garlan, J. Ivers, R. Little, R. Nord, and J. Stafford, Documenting Software Architectures: Views and Beyond. 25th International Conference on Software Engineering (ICSE'03), 2003.

[CGY03] Lawrence Chung, D. Gross, and Eric Yu, Architecture Design to Meet Stakeholder Requirements, From Software Requirements to Architectures Workshop (STRAW), May 2003.

[CNY94] L. Chung, B. A. Nixon and E. Yu, "Using Quality Requirements to Systematically Develop Quality Software," Proc., 4th International Conference on Software Quality, McLean, VA, U.S.A. Oct. 3-5, 1994.

[CNY95a] L. Chung and B. Nixon and E. Yu, "Using Non-Functional Requirements to Systematically Support Change," Proc., IEEE 2nd International Symposium on Requirements Engineering, York, England, March 27-29, 1995., pp. 132-139.

[CNY95b] L. Chung, B. Nixon and E. Yu, "Using Non-Functional Requirements to Systematically Select Among Alternatives in Architecture Design," Proc., 1st International Workshop on Architectures for Software Systems, Seattle, April 24-28, 1995., pp. 31-43.

[CNYM00a] Lawrence Chung, Brian A. Nixon, Eric Yu, and John Mylopoulos. Quality-attribute Requirements in Software Engineering. Springer, 2000.

[CNYM00b] L. Chung, B.A. Nixon, E. Yu, J. Mylopoulos, "Non-Functional Requirements in Software Engineering", Kluwer Academic Publishers, 2000. ISBN 0-7923-8666-3.

[DN02] Liliana Dobrica, and Eila Niemela, A Survey on Software Architecture Analysis Methods. IEEE Transactions on Software Engineering, vol. 28, No. 7, July 2002.

[Fox06] C. Fox, Introduction to Software Engineering Design: Processes, Principles and Patterns with UML2. Addison Wesley, 2006.

[GB01] P. Grunbacher, B. Boehm, "EasyWinWin: a groupware-supported methodology for requirements negotiation", Proceedings of the 8th European Software Engineering Conference held jointly with 9th ACM SIGSOFT international symposium on Foundations of Software Engineering, Austria, 2001.

[GHJV95] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, Design Patterns: Elements of Reusable Object-Oriented Software, Addison Wesley, 1995.

[GMNS04] P. Giorgini, J. Mylopoulos, E. Nicchiarelli, and R. Sebastiani, "Formal Reasoning Techniques for Goal Models", Journal on Data Semantics 1: 1-20, 2004.

[GY01] D. Gross, E. Yu, "From Non-Functional Requirements to Design through Patterns", Requirements Engineering. Springer-Verlag. 6(2001) 1: 18-36.

[HK94] Raimo P. Hämäläinen and Eero Kettunen, On-line Group Decision Support by HIPRE 3+ Group Link, Proceedings of the Third International Symposium on the Analytic Hierarchy Process, Washington, D.C., July 11-13, 1994.

[HKC00] Rick Hazman, Mark Klein, Paul Clements, "ATAM: Method for Architecture Analysis", CMU/SEI-2000-TR-004, Software Engineering Institute, Carnegie Mellon University, 2000.

[IMP01] P. Inverardi, H. Muccini and P. Pelliccione. Automated Check of Architecture Models Consistency using SPIN. IEEE Proc. Automated Software Engineering conference (ASE'01), 2001.

[JW98] D. Jennings and S. Wattam, "Decision Making: an Integrated Approach", London, Washington D.C., Financial Times Pitman Pub, 1998.

[Ka96] R. Kazman, "Tool Support for Architecture Analysis and Design", Proceedings of the 2nd International Software Architecture Workshop, San Francisco, California, 1996.

[KAK01] R. Kazman, J. Asundi, and M. Klein, Quantifying the Costs and Benefits of Architecture Decisions, Proceedings of the 23rd International Conference on Software Engineering (ICSE 23), (Toronto, Canada), May 2001, 297-306.

[KBAW96] Kazman, R., Bass, L., Abowd, G., Webb, M., "SAAM: A Method for Analyzing the Properties of Software Architectures", Proceedings of ICSE 16, Sorrento, Italy, May 1994, 81-90.

[KCW00] R. Kazman, S.J. Carriere, and S.G. Woods, Toward a Discipline of Scenario-Based Architecture Engineering, Annals of Software Engineering, vol. 9, 2000.

[KIC05] R. Kazman, H. In, H. Chen, "From Requirements Negotiation to Software Architecture Decisions", Information and Software Technology, Vol. 47, Issue 8, June 2005.

[KK99] M. Klein and R. Kazman, "Attribute-based Architecture Styles", Tech. Report, CMU/SEI-99-TR-022, Software Engineering Institute, Carnegie Mellon University.

[KKC00] R. Kazman, M. Klein, and P. Clements, ATAM: Method for Architecture Evaluation, Technical Report, CMU/SEI-2000-TR-004.

[KKC04] Donald L. Keefer, Craig W. Kirkwood, James L. Corner, Perspective on Decision Analysis Applications, Decision Analysis, vol. 1(1), 2004

[Kl93] M.H. Klein, et al., "A practitioner's Handbook for Real-time Analysis: Guide to Rate Monotonic Analysis for Real-time Systems", Kluwer Academic, 1993.

[KR76] Ralph Keeney and Howard Raiffa, Decisions with Multiple Objectives: Preferences and Value Tradeoffs. New York: Wiley

[Kru95] P.B. Krutchen, "The 4+1 View Model of Architecture", IEEE Software, pp.42-50. Nov.1995.

[Lam03] A. van Lamsweerde, From System Goals to Software Architecture, SFM 2003.

[LBKK97] C. Lung, S. Bot, K. Kalaichelvan, and R. Kazman, "An Approach to Software Architecture Analysis for Evolution and Reusability", Proceedings CASCON'97, Nov. 1997.

[LK95] D.C. Luckham, J.J. Kenney, et al. Specification and Analysis of System Architecture Using Rapide. IEEE Transactions on Software Engineering. 21(4), p. 336-355, April, 1995.

[LL02] P. Leviakangas, J. Lahesmaa, Profitability evaluation of intelligent Transport System Investments, Journal of Transportation Engineering-ASCE 128(3): 276-286, May-Jun 2002.

[LTW99] V.S. Lai, R.P. Trueblood, B.K. Wong, Software Selection: a Case Study of the Application of the Analytical Hierarchy Process to the Selection of a Multimedia Authoring System, Information & Management, Vol 36, Nov. 4, 1999.

[Ly96] M.R. Lyu, ed. "Handbook of Software Reliability Engineering", McGraw-Hill and IEEE Computer Society: New York, 1996.

[Mat00] Mattingly, Stephen P., Decision Theory for Performance Evaluation of New Technologies Incorporating Institutional Issues: Application to Traffic Control Implementation, PhD Dissertation, University of California, Irvine, CA.

[MCN92] J.Mylopoulos, L. Chung, and B. Nixon, "Representing and Using Non-Functional Requirements: A process-Oriented Approach";, IEEE Transactions on Software Engineering, Special Issue on Knowledge Representation and Reasoning in Software Development, 18(6), June 1992, pp. 483-497.

[MDR04] H. Muccini, M. Dias, and D.J. Richardson, Systematic Testing of Software Architectures in the C2 Style, 7th International Conference on Fundamental Approaches to Software Engineering (FASE 2004), Springer, 2004.

[Min92] H. Min, Selection of Software: The Analytic Hierarchy Process, International Journal of Physical Distribution & Logistic Management, 1992.

[MJM+99] Moore, James E., II, R. Jayakrishnan, Michael G. McNally and C. Arthur MacCarley with Hsi-Hwa Hu, Steve Mattingly, Seongkil Cho and James Roldan. Evaluation of the Anaheim Advanced Traffic Control System Field Operational Test: Introduction and Task A: Evaluation of SCOOT Performance. Final report to Federal Highway Administration, California PATH Research Report UCB-ITS-PRR-99-26, July 1999.

[MJM00a] S.G. Mattingly, R. Jayakrishnan, and M.G. McNally, Determining The Overall Value of Implemented New Technology in Transportation: Integrated Multiple Objective-Attribute Methodology, Transportation Research Record No. 1739, 2000.

[MJM00b] Stephen P. Mattingly, R. Jayakrishnan, Michael G. McNally (2000), Decision Theory for the Performance Evaluation of a New Traffic Control System, Proceedings of the International Conference on Decision Making in Urban and Civil Engineering, Lyon, France, Nov 2000.

[MMM+99] McNally, Michael G., James E. Moore, II, C. Arthur MacCarley and R. Jayakrishnan, Steve Mattingly, James Roldan, Hsi-Hwa Hu and Seongkil Cho. Evaluation of the Anaheim Advanced Traffic Control System Field Operational Test: Task B: Assessment of Institutional Issues. Final report to Federal Highway Administration, California PATH Research Report UCB-ITS-PRR-99-27, July 1999

[NWC04] R.L. Nord, W.G. Wood, P.C. Clements, "Integrating the quality attribute workshop and the Attribute-Driven Design Method", technical report, CMU/SEI-2004-TN-017.

[Par72] D. Parnas. On the criteria to be Used in Decomposing Systems into Modules. Comm. ACM, vol. 15, no. 12, pp. 1053-1058, 1972.

[PW92] Dewayne E. Perry, and Alexander L. Wolf. Foundations for the Study of Software Architecture. ACM Software Engineering Notes 17(4): 40-52, 1992.

[PZJ07] Ji Young Park, Yu Zhang, R. Jayakrishnan, Conceptual Framework with Real-Time Distributed Vehicle Data for Traffic Control, Paper 07-3399, Proc. Transportation Research Board Annual Meeting, Washington DC, 2007

[Saa82] T.L. Saaty, Decision Making for Leaders: The analytical hierarchy process for decisions in a complex world, Wadsworth, Belmont, CA, 1982.

[SW93] C.U. Smith and L.G. Williams, "Software Performance Engineering: A Case Study Including Performance Comparison with Design Alternatives", IEEE Transactions on Software Engineering, 1993. 19(7).

[SWLM02] M. Svahnberg, C. Wohlin, L. Lundberg, and M. Mattsson, "A Method for Understanding Quality Attributes in Software Architecture Structures", SEKE'02, July 15-19, 2002, Italy.

[SWLM03] Mikael Svahnberg, Glaes Wohlin, Lars Lundberrg, and Michael Mattsson, "A Quality-Driven Decision-Support Method for Identifying Software Architecture Candidates", International Journal of Software Engineering and Knowledge Engineering, Vol. 13, No. 5, 2003.

[UCKM04] Sebastian Uchitel, Robert Chatley, Jeff Kramer and Jeff Magee, System Architecture: the Context for Scenario-based Model Synthesis, SIGSOFT 2004/FSE12, Newport Beach, CA.

[VDR00] Marlon Vieira, Marcio Dias, and Debra Richardson, Analyzing Software Architectures with Argus-I. Proceedings of the International Conference on Software Engineering (ICSE 2000). p. 758-761, Limerick, Ireland, June 4-11, 2000.

[WG94] Wesseling, J A M and Gabor, A, Decision Modelling with HIPRE 3 Plus: The Amsterdam Airport Case, Computational Economics, Springer, vol. 7(2), pages 147-54

[XHH+06] L. Xu, S.A. Hendrickson, E. Hettwer, H. Ziv, A. van der Hoek, and D.J. Richardson, Towards Supporting the Architecture Design Process through Evaluation of Design Alternatives, Second International Workshop on the Role of Software Architecture for Testing and Analysis, July 2006, pages 38–44.

[XZAR06] Lihua Xu, Hadar Ziv, Thomas A. Alspaugh and Debra J. Richardson, An architecture pattern for quality-attribute dependability requirements, Journal of Systems and Software, October 2006, Volume 79, issue 10, Special Issue on Architecting Dependable Sytems.

[ZAGJ05] Liming Zhu, Aybuke Aurum, Ian Gorton, and Ross Jeffery, "Tradeoff and Sensitivity Analysis in Software Architecture Evaluation Using Analytical Hierarchy Process", Software Quality Journal , 13, 357-375, 2005.

[ZMR+07] Yu Zhang, James E. Marca, Craig Rindt, R. Jayakrishnan, Michael G. McNally, Efficient Path and Subpath Storage Data Structures for Network Travel Analysis with Route Behavioral Elements, Paper 07-3029, Proc. Transportation Research Board Annual Meeting, Washington DC, 2007.