Carnegie Mellon
Software Engineering Institute

Using Economic Considerations to Choose Among Architecture Design Alternatives
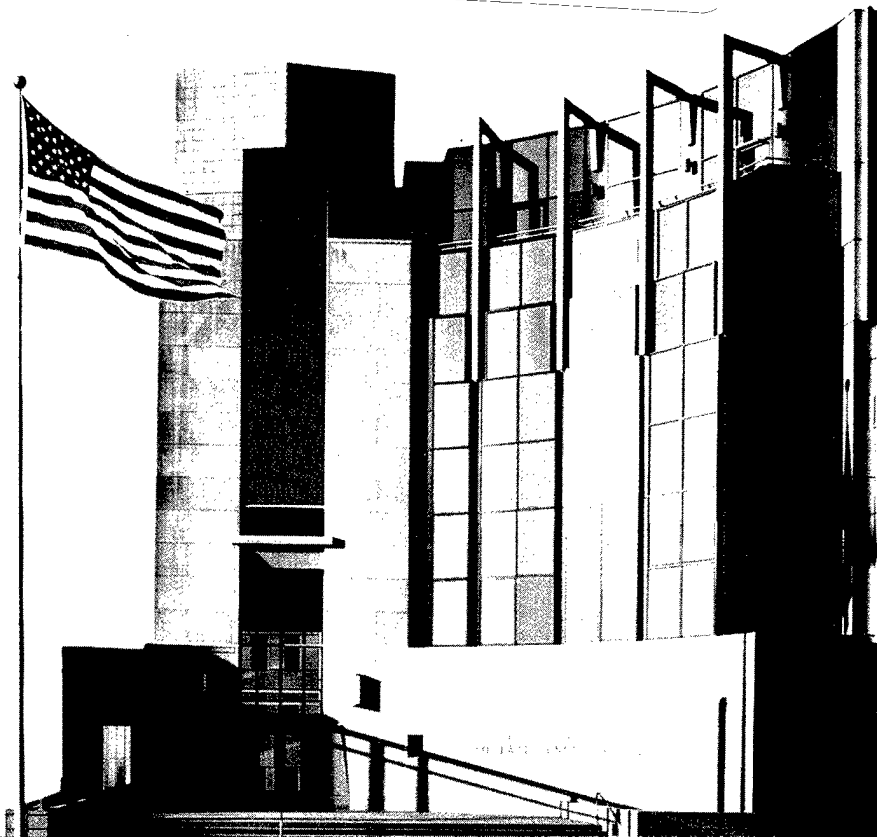
Jayatirtha Asundi
Rick Kazman
Mark Klein

December 2001

20020221 021

TECHNICAL REPORT
CMU/SEI-2001-TR-035
ESC-TR-2001-035

Carnegie Mellon
Software Engineering Institute

Pittsburgh, PA 15213-3890

Using Economic Considerations to Choose Among Architecture Design Alternatives

CMU/SEI-2001-TR-035
ESC-TR-2001-035

Jayatirtha Asundi
Rick Kazman
Mark Klein

December 2001

Product Line Systems
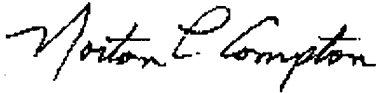
This report was prepared for the

The ideas and findings in this report should not be construed as an official DoD position. It is published in the interest of scientific and technical information exchange.

FOR THE COMMANDER

Norton L. Compton

Norton L. Compton, Lt Col, USAF
SEI Joint Program Office

Table of Contents

List of Figures

List of Tables

Abstract

The software architecture forms an essential part of a complex software-intensive system. Architecture design decision-making involves addressing tradeoffs due to the presence of economic constraints. The problem is to develop a process that helps a designer choose amongst architectural options, during both initial design and its subsequent periods of upgrade, while being constrained to finite resources. To address this need for better decision-making, we have developed a method for performing economic modeling of software systems, centered on an analysis of their architecture. We call this method the Cost Benefit Analysis Method (CBAM). The CBAM incorporates the costs and benefits of architectural design decisions and provides an effective means of making such decisions. The CBAM provides a structured integrated assessment of the technical and economic issues and architectural decisions. The CBAM utilizes techniques in decision analysis, optimization, and statistics to help software architects characterize their uncertainty and choose a subset of changes that should be implemented from a larger set of alternatives. We also report on the application of this method to a real world case study.

1 Introduction and Motivation

The software architecture is an essential part of a complex software-intensive system. Shaw and Garlan [Shaw 96] state that, with increasing complexity of a system, the specification of the overall system, i.e., its software architecture, becomes a more significant issue than the choice of algorithms or data structures. The Architecture Tradeoff Analysis Method (ATAM) [Kazman 00] provides software architects a framework to reason about the technical tradeoffs faced while designing or maintaining a software system. In the ATAM, we are primarily investigating how well the architecture has been designed with respect to its quality attributes (QAs) that include modifiability, performance, availability, and usability. It is these qualities that shape the architecture and consequently dictate the cost of building and maintaining a software system. The ATAM also analyzes architectural tradeoffs, the places where a decision might have consequences for several QA concerns simultaneously.

The biggest tradeoffs in large, complex systems usually have to do with economics. How should an organization invest its resources in a manner that will maximize its gains and minimize its risk? Where this question has been addressed, it has primarily focused on costs [Boehm 81], and even then these costs are primarily the costs of building the system in the first place, and not its long-term costs through cycles of maintenance and upgrade. The benefits that an architectural decision may bring to an organization are just as important as the costs.

Given that the resources for building and maintaining a system are finite, there must be a rational process that helps us choose amongst architectural options, during both initial design and its subsequent periods of upgrade. These options will have different costs, will implement different features, each of which brings some benefit to the organization, and will have some inherent risk or uncertainty. Thus, we need economic models of software that take into account costs, benefits, risks, and schedule implications.

To address this need for better economic decision-making, we have developed a method for performing economic modeling of software systems, centered on an analysis of their architecture. We call this method the CBAM (Cost Benefit Analysis Method). The CBAM builds upon the ATAM to model the costs and the benefits of architectural design decisions and provides a means of optimizing such decisions. The CBAM provides a structured integrated assessment of the technical and economic issues and architectural decisions. In our framework, we incorporate economic criteria like benefit and cost that are derived from the technical criteria like quality attribute responses.

The following section describes the decision-making context behind the CBAM. while section 3 provides a broad outline of the steps involved. In section 4 we describe the steps in detail as well as the theoretical basis of the CBAM. We apply the method to a real world project, NASA's Earth Observatory System Distributed Information System (EOSDIS) Core System (ECS), which is outlined as our case study in section 5. We describe related work in section 6, discuss our plans for improving the method in section 7, and finally conclude in section 8.

2 Decision-Making Context

In the CBAM the software architect or decision-maker wishes to maximize the difference between the benefit he or she derives from the system, and the cost of implementing the design. The CBAM begins where the ATAM concludes, and in fact, depends upon the artifacts that the ATAM produces as output. Figure 1 depicts the context for the CBAM. The business goals of a software system are expected to influence the architecture decisions made by software architects or designers. These architecture decisions have economic as well as technical implications. The direct economic implication is that of the cost of building/implementing the system. The technical implications are the characteristics of the software system, namely the quality attributes. These quality attributes in turn have economic implications for the benefit that can be derived from the system.
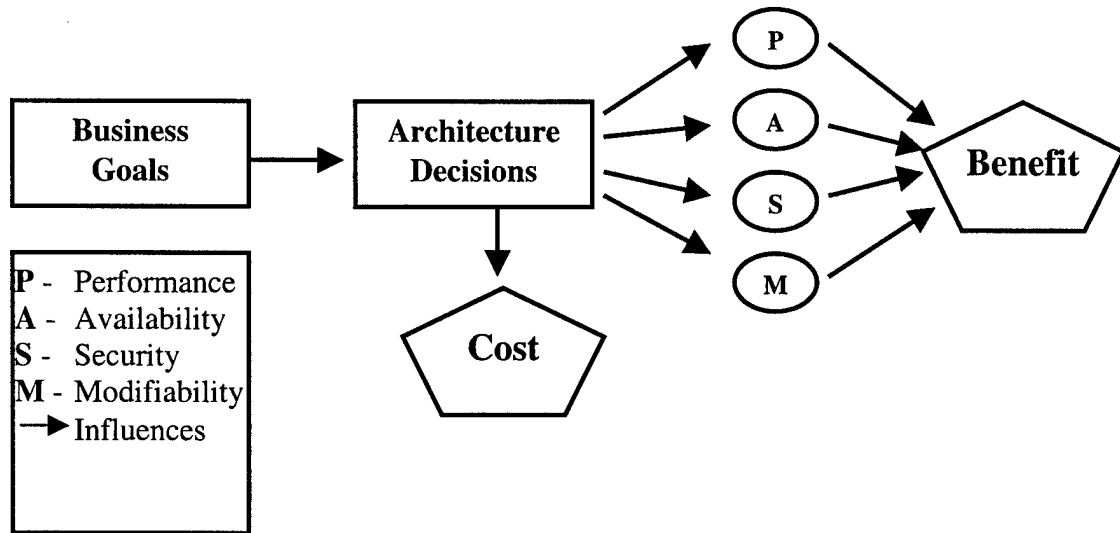
Figure 1: Context of the Cost Benefit Analysis Method (CBAM)

When the ATAM is applied to a software system, we expect to have a set of artifacts documented upon completion. They are as follows:

- a description of the business goals that are crucial to the success of the system

- a set of architectural views that document the existing or proposed architecture

- a utility tree that represents a decomposition of the stakeholders' goals for the architecture. The utility tree starts with high-level statements of QAs, decomposes these into spe-

cific instances of QA requirements (performance, availability, etc.) and realizes these as scenarios

- a set of risks that have been identified

- a set of sensitivity points (architectural decisions that affect some QA measure of concern)

- a set of tradeoff points (architectural decisions that affect more than one QA measure, some positively and some negatively)

The ATAM results in a set of key potentially problematic architectural decisions, based on QA scenarios elicited from the stakeholders. These architectural decisions result in some specific QA responses, namely, a particular level of performance, security, usability, modifiability, and so forth. But those architectural decisions also have an associated cost. For example, if an architectural decision is made to use redundant hardware to increase reliability, then this has one cost consequence. If check pointing to a disk file is used instead, then this architectural decision has a different cost. Furthermore, both of these architectural decisions will result in a measurable level of reliability (perhaps measured as mean time to failure or steady-state availability). These QA responses will have some value to the organization developing software. Perhaps the organization believes that its stakeholders will pay extra for a highly reliable system (a telephone switch, for example) or that the organization will get sued if the system is not highly available (for example, a medical monitoring device).

The ATAM uncovers the architectural decisions made and links them to business goals and QA response measures. The CBAM builds on this foundation by filling in the pentagons in Figure 1, by eliciting the *costs* and *benefits* associated with these decisions.[1] Given this information, the stakeholders can then decide whether to use redundant hardware, check pointing, or some other architectural decision addressed at increasing the system's reliability. Or the stakeholders can choose to invest their finite resources in some other QA—perhaps believing that higher performance will have a better benefit to cost ratio. A system always has a limited budget for creation or upgrade, so every architectural choice is, in some sense, competing with every other one for inclusion. The problem addressed in this technical report is that, given the inevitable constraints on cost, how should an architect choose a small set of architectural strategies to be implemented from a large set of alternatives?

The CBAM does not make decisions for the stakeholders; it simply aids them in the elicitation and documentation of costs, benefits, and uncertainty and gives them a framework within which they can apply a rational decision-making process.

---

[1]   The CBAM does not include a new way of determining costs (although we think that an architecture-aware cost estimation method is a desirable goal). It assumes that some method of cost estimation already exists within the organization.

3 The Steps of the CBAM

The CBAM consists of six steps.

1. Choose the scenarios and architectural strategies (ASs).

2. Assess the relative importance of QAs.

3. Quantify the benefit scores of the ASs.

4. Quantify the costs of the ASs and incorporate schedule implications.

5. Calculate the 'return' for each AS.

6. Rank order the ASs and apply an appropriate decision rule.

When we start with a CBAM exercise, there is a large number of 'desired' architectural changes or ASs. Performing a detailed CBAM exercise involving exhaustive elicitation and analysis of each of these changes is an impossible task. To wade through such a large space of possible changes makes it necessary to use a two-phase approach. Hence, the CBAM consists of two phases: the triage phase, where rough order of magnitude estimates are used to prune the decision space, and the detailed examination phase, where more detailed estimates are gathered about the more promising architecture design alternatives. The six steps, described above, are carried out in both phases.

3.1 The Triage Phase

In the triage phase of the CBAM we elicit the costs and benefits of the changes in a rough qualitative manner. Another step is to find a consensus amongst the stakeholders about the definition of the various QAs of importance as elicited during the ATAM. This exercise is carried out so that the stakeholders have a common understanding about the attributes that they are going to use as a basis for rating the various ASs.

Table 1: Elicited Triage Information

Architectural Strategy	Performance (50)	Availability (30)	Modifiability (20)	Cost
AS1	++	+	--	L
AS2	+	0	--	M
AS3	-	0	++	H
AS4	0	+	++	H
AS5	++	-	0	M
AS6	--	+	++	L

During triage, we request the experts to rate every AS by QA on a five-point scale (--, -, 0, +, ++). For the cost estimates we use a simple three-point scale: low, medium, high. The various QAs are assigned votes, on the basis of their importance to the system, such that the sum of the votes is equal to 100. An example of the elicited information is shown in Table 1. The qualitative scales are converted to rough quantitative estimates for benefit as well as cost. The ASs are then plotted on a graph of benefit against cost as shown in Figure 2. This exercise helps in visualizing the alternatives.
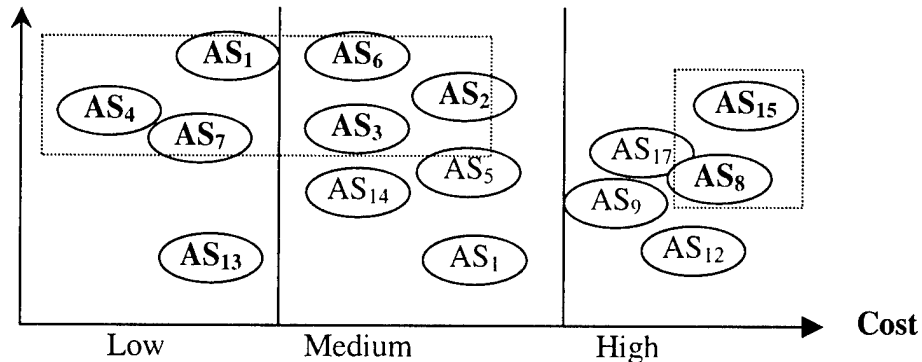
Figure 2: ASs Mapping on a Benefit-Cost Plot During Triage Phase[2]

The ovals surrounding the ASs depict the uncertainty region. The chosen ASs are shown in bold. The number of ASs considered for detailed analysis are pruned on the basis of having a high benefit - low cost characteristic, other factors like regulations/standards and that they have no resource conflicts or dependencies. In this manner, triage is used to prune the set of ASs to a manageable number of alternatives that can be analyzed in detail. We expect about 20-25 ASs for detailed examination.

It is quite possible that some organizations may not feel the need for a detailed analysis due to resource constraints. For some, order of magnitude estimates or qualitative measures are sufficient for their decision-making process. In these cases the triage could prove to be sufficient for their needs.

3.2 The Detailed Examination Phase

In the detailed examination phase we perform a detailed analysis on the 20-25 ASs that were found to be promising in the triage phase. The strategies are re-assessed for their impact on the software system's QAs, and a more quantitative scale, which varies from -1 to +1, replaces the -/+ scale of the triage phase. We describe the theoretical basis and the step details in the following section.

---

[2] The ovals shown within each cost category are representative and not actual positions.

4 Theoretical Basis of the CBAM and Detailed Description

The architecture design problem of analyzing tradeoffs between the various quality attributes of a system can be characterized in a traditional multi-attribute decision theory framework. We use this framework as the basis for the CBAM. We explore the problem of uncertainty [Morgan 90] and also examine techniques like portfolio analysis [Markowitz 52] to tackle our problem. Multi-attribute decision theory as described by Keeney and Raffia [Keeney 76] is adapted to software systems, and the formalisms are presented in Appendix A.

Complex software decision problems involve conflicting objectives with multiple attribute measures. Usually, there is no alternative that is superior, in all respects, to other alternatives, i.e., no dominating solution. There are value tradeoffs between the various quality attributes of the system. For example, one alternative may have high performance but low maintainability, while another alternative has high maintainability but low performance. Choosing between the two alternatives will involve understanding the value judgments of the key stakeholders in the software project. (A dominating alternative would be one with high performance and high maintainability—but rarely do we find such easy and obviously optimal solutions to problems!)

The decision-making framework thus involves

1. determining the scaling parameters or the relative importance of the attributes (which we call $QAScore_j$)

2. determining the attribute-specific impact or utility values (which we call $ContribScore_{i,j}$.)

The product of these values gives us a measure of the benefit due to an AS and can thus be used as an evaluation metric. The details are described in the following sections.

4.1 Step 1: Choose Scenarios and Architectural Strategies (ASs)

In the first step, we choose a smaller number of ASs for detailed analysis. Frequently it is the case that certain changes are dictated by external forces—keeping up with a competitor, being forced to port to newer hardware, meeting government regulations, complying with standards, etc. Thus some of the ASs considered in the triage phase may be automatically consid-

ered in the next phase in spite of their benefit/cost value. These are indicated by AS_8 and AS_{15} in Figure 2. From the triage phase, we also identify the ASs that have a high benefit but low to medium cost; this is shown by the dotted rectangle in Figure 2. Some ASs may be excluded due to resource conflicts or lack of support from the management. Additionally, some ASs that are outside this region may be included. because of dependencies (i.e., AS_{13} may depend on AS_7). A few other ASs may be chosen for analysis due their proximity to already chosen ASs (like AS_5 and AS_{14}). Thus a smaller set of ASs are chosen for detailed analysis.

4.2 Step 2: Assess the Relative Importance of QAs (Elicit QAScore$_j$)

Once the various quality attributes of importance have been identified, the scaling parameter QAscore$_j$ for each quality attribute, j, is elicited. By design, we ensure that

$$\sum_j QAScore_j = 100$$

$$\text{and } \forall j : QAScore_j \geq 0$$

An example of elicited QAscore parameters is shown in Table 2.

Table 2: Example of Scaling Parameters Elicited from Stakeholders

Quality Attribute	Stakeholder 1	Stakeholder 2	Stakeholder 3
Attribute1	50 (1)	60 (1)	40 (1)
Attribute2	15 (3)	15 (2)	15 (3)
Attribute3	25 (2)	10 (4)	30 (2)
Attribute4	10 (4)	15 (2)	15 (3)
$\sum$QAscore	100	100	100

The stakeholders are usually from the same organization or have similar understanding of the goals of the system. They are frequently in accordance with each other regarding the scaling parameters. However, there are times when they may show great differences of opinion. This difference in opinion will manifest as a high variability of elicited values. The variability could be either due to the inherent uncertainty present in the system or that the stakeholders are not completely in agreement with each other regarding the definition of the scaling parameters. To test how much the stakeholders agree with each other regarding the scaling parameters, we use the Kendall's[3] concordance coefficient as a measure of agreement for the group as a whole [Kendall 55]. The more the group agrees over the rank of each attribute, the higher the concordance coefficient.

In the example shown in Table 2, we have 3 stakeholders and 4 attributes. The values in the columns show the elicited scaling parameters and the numbers in the bracket show the rank

---

[3]   It is similar to Spearman's correlation coefficient but it is a non-parametric measure.

of that particular attribute for that particular stakeholder. Kendall's concordance coefficient reflects how much the stakeholders agree on the rank of the particular attribute. In this example, the Kendall's coefficient of concordance, W, is 0.6905 (F-stat = 4.461, p = 0.08), which shows a fair level of concordance, though it is not significant. Though the values elicited from the stakeholders may differ, a consistent rating on the relative importance of each of the attributes shows that the stakeholders more or less agree as to which attributes are important to the business goals of the system. If the concordance value is low or not significantly greater than zero, then the stakeholders will need to revisit the business goals of the system and definitions of the QAs. This is to ensure that the stakeholders have a common understanding of the QAs and their respective implications on the business goals of the system. This iteration is carried out to ensure that variability due to lack of understanding is reduced.

4.3 Step 3: Quantify the Benefits of ASs (Elicit ContribScore$_{i,j}$)

A single architectural strategy affects more than one QA. Some effects are positive, some negative, and all to varying degrees. When considering large software systems, there is uncertainty regarding the exact effect of a particular architectural strategy on the system. While the effect of some changes could be measured approximately through simulation models, the effect of some changes can only be obtained through expert elicitation of the experts of the system and their understanding of system behavior. Another source of uncertainty and subjectivity is the utility assessment of various levels of the quality attributes. (See Figure 7 in Appendix A.)

In this step we ask the experts to rate each architectural strategy (AS$_i$) in terms of its contribution (ContribScore$_{i,j}$) to each quality attribute (QA$_j$) on a scale of -1 to $+1$. A plus 1 implies that the AS has a perceived best possible effect on the QA for the system and a minus 1 means the opposite. These values represent the utility gained or lost by making a particular architectural change (AS$_i$) on each of the quality attributes (QA$_j$).

4.3.1 Addressing Variability in the Contribution Score

So far we have explained a deterministic case of eliciting contribution scores. However, with multiple stakeholders and sources of uncertainty the elicited values will be ranges and not single point estimates. As explained in the previous section, we assume that the variability in contribution scores may be due to two following reasons:

1. the existence of uncertainty in the system response characteristics or

2. a difference in interpretation or understanding of the system or particular subsystems by experts

We address the variability due to uncertainty in our analysis of elicited values explained later in this report. However, the variability arising due to the interpretation or understanding of

the system is something we wish to minimize (if not eliminate completely). Here we use the Delphi method [Linstone 75] for obtaining consensus on the specific impacts of particular architectural strategies on the system. The specific process we use is the following:

1. Obtain the first round of contribution scores from all experts including a short one-line statement as to why the expert has rated a particular AS in that way. For example, the expert could annotate the comment: *"Improved logging would allow systematic security audits"* against the rating for security.

2. Also obtain self-evaluations of the experts, with regard to their understanding of the impacts of the AS, on a four point scale ('Don't have any idea,' 'have some idea,' 'have a reasonable idea,' and 'have a good idea').

3. Compile scores to produce ranges and the comments for each AS.

4. Provide all the experts with the range and comment information and ask them to review their ratings based on the collected information. We do not reveal the expert raters' identity to reduce the possibility of influencing other raters.

5. Obtain a final range on contribution scores and compile the one-line reasons given by experts as documentation.

In this manner, we expect to minimize the variability of elicited values due to lack of information or lack of understanding of all the implications of a change. We also give the experts the choice to recuse themselves from rating certain architectural changes due to their lack of understanding about the impacts of an AS.

The measure of variability in the scores we use is the Kendall's concordance coefficient. We compute this coefficient from the elicited contribution scores and use it as an indicator of how consistent the raters are with each other. A low coefficient indicates that the variability in ratings is high and it is plausible that some experts may be missing certain information that other experts possess. Thus, we could use the coefficient as a measure of the stopping point for the iterative Delphi process of circulating information and eliciting ratings.

Though this process seems tedious and time consuming, we feel that this is necessary to increase the objectivity of the ratings and make sure that every expert has the same information to make the rating. Another benefit of this exercise is to capture the information contained in the one-line statements made by the experts. This may be used to build models of the system and to generate utility-attribute curves. (See Figure 8 in Appendix A.)

4.3.2 Calculating the Benefit Scores of the ASs

At this point in the method we have elicited values for the scaling parameter, $QAscore_j$, to reflect the importance of the QAs with respect the business goals and the utility of each architectural change, $ContribScore_{i,j}$. We can now calculate the benefit score for each architectural strategy by the following formula:

$$Benefit(AS_i) = \sum_j (ContribScore_{i,j}) \times (Qattrib_j)$$

Since the ContribScore is a utility estimate, the benefit score is expressed in 'utils.' For example, consider a software system with the following scaling parameters:

(performance, security, availability, modifiability) = (25, 30, 15, 30)

and the rating of the architectural strategies for the respective QAs as follows:

AS_1 (1, -0.5, 0.6, -0.4)

AS_2 (-0.4, 1.0, 0.8, 1.0)

The respective benefit scores will be:

Benefit(AS_1) = (1)*25 + (-0.5)*30 + (0.6)*15 +(-0.4)*30 = 7 utils

Benefit(AS2) = (-0.4)*25 + (1)*30 + (0.8)*15 +(1.0)*30 = 71.2 utils

This benefit score utility estimate is bounded between -100 and +100 utils.

4.4 Step 4: Quantify the Costs of the AS and Incorporate Schedule Implications

We have described in detail a method to elicit benefits of various architectural strategies. There are few studies that discuss the *benefit* of architectural strategies. Cost estimation on the other hand, as far as the implementation cost is concerned, has received considerable attention in the software engineering literature [Boehm 88, Jones 99]. Typically, most mature organizations adopt their own methods for cost estimation across all projects. The CBAM does not include any new way of determining costs. However, considering that most cost estimation techniques are dependent on much finer detailed parameters like lines of code or number of variables and other implementation details, we think that an "architecture-aware" cost estimation method is a desirable goal. This architecture-aware cost model would enable us to obtain cost estimates based on the type of components used or the architectural styles used to design the system.

As of now, the CBAM assumes that some method of cost estimation already exists within the organization, even if it is ad-hoc, and we can directly obtain cost estimates, C_i, for each of the architectural strategies. In addition to eliciting the costs and benefits of the ASs under consideration, prudent planning dictates that we estimate the schedule implications of each AS_i in terms of elapsed time, shared use of critical resources, and dependencies among implementation efforts. Perhaps an AS is otherwise desirable, but does not fit in with the organization's time-to-market goals. During this step we will note any contention for shared resources among these estimates (hardware, software, or personnel), for these will also affect the feasibility of an AS.

4.5 Step 5: Calculate Return for Each AS

The next step in the process of ranking our architectural strategies is to calculate the return score (r_i). This score is given by

$$\text{Return }(AS_i) = \frac{\text{Benefit }(AS_i)}{\text{Cost }(AS_i)}$$

The units of the return score will differ according to the units of benefit and cost elicited. Ideally the return score will be a non-dimensional number since both the benefit and cost are utility values, in utils or in dollars. Due to the range of values in benefit as well as cost, the calculated return scores have a range of values. The return score is similar to a return-on-investment (ROI) metric used commonly in the financial literature.[4]

4.6 Step 6: Rank Order the ASs and Apply an Appropriate Decision-Rule

A decision rule that is based on the mean value of the return could be applied. The ASs could be rank ordered according to the mean value of their return. This assumes that the underlying distribution of the return is symmetric around the mean value. The top 'r' ASs could then be chosen such that

$$\sum_{i=1}^{r} Cost\,(AS_i) \le K$$

where K is the total budget amount available for the project.

The number of stakeholders is usually small and the mean value could be skewed considerably because one stakeholder varies considerably from other stakeholders. To account for this, another criterion for rank ordering is the median value of the return score. Similarly, the top 'r' ASs could be chosen, subject to the cost constraint as shown above.

4.6.1 A Decision-Rule Based on Probability

In the above two ranking methods, we have assumed that the central moments, namely the mean and median estimates for return, accurately represent the AS. Using the mean or median values of return assumes that the underlying distribution of the return score is normal/log-normal. Considering that uncertainty is fairly high, and the underlying distribution at times skewed, an expected value decision rule may not be sufficient to capture the uncertainty. In this section we briefly describe a technique to incorporate the uncertainty into our decision.

---

[4]   The relation between our defined return and ROI is: ROI = (return − 1) x 100 if the units of cost and benefit are the same.

Considering that the number of stakeholders is usually small and each stakeholder conveys important information about the elicited values, we assume that the underlying distribution of the return score is uniform. This assumption is reasonable considering that the outlier may be examining the system from a perspective that other stakeholders may not be able to appreciate. If each architectural strategy is plotted along a single scale, according to its return score, there could be overlap of scores between any two ASs.

Given that the values overlap, we cannot say for certain which AS is preferable. For this purpose we develop a probabilistic technique, where we calculate the probability that the return score of any AS_i is greater than the return of another AS_j. The detailed derivations for two cases of possible overlap are provided in Appendix B.

Case: 1: When there is partial overlap between AS_i and AS_j.

$$prob\left(AS_i \geq AS_j\right) = \frac{1}{2R_iR_j}\left\{AS_{i,max} - AS_{j,min}\right\}^2$$

Case: 2: When AS_i completely overlaps AS_j.

$$prob\left(AS_i \geq AS_j\right) = \frac{1}{2R_i}\left\{2AS_{i,max} - AS_{j,max} - AS_{j,min}\right\}$$

where R_j is the total range of the return value for AS_j and $AS_{j,min}$ and $AS_{j,max}$ are the minimum and maximum return values of AS_j respectively.

For example, consider a sample set of 6 ASs. The above technique yields the probability matrix shown in Table 3. Each value in the matrix denotes the Probability (Row-AS >= Column-AS).

Table 3: Probability of $AS_{row} > AS_{column}$

	AS1	AS2	AS3	AS4	AS5	AS6	Avg.
AS1	0.5	0.3	0.6	0.7	0.4	0.9	0.57
AS2	0.7	0.5	0.7	1.0	0.6	1.0	0.75
AS3	0.4	0.3	0.5	0.6	0.3	0.7	0.47
AS4	0.3	0.0	0.4	0.5	0.2	0.3	0.28
AS5	0.6	0.4	0.7	0.8	0.5	0.6	0.60
AS6	0.1	0.0	0.3	0.7	0.4	0.5	0.33

The probabilities in Table 3 are used to determine the sensitivity of our earlier ranking decision. If we choose AS5 over AS6, then the table tells us that there is a 40% probability that we may be incorrect in our rank ordering. If we fix p >= 0.6, as a limit for partial dominance, then we get the following result:

AS1 > AS3, AS4, AS6

AS2> AS1, AS3, AS4, AS5, AS6 (Clearly a highly ranked AS)

AS3> AS4, AS6

AS4>

AS5> AS1, AS3, AS4, AS6

AS6>AS4

We can see that in these conditions, AS2 is the most preferred AS, followed by AS5, AS1, AS3 AS6, AS4 in that order.

We use this probabilistic framework to rank the different architectural strategies. We have shown how a decision rule based on a particular probability value, 'p', can be used to incorporate the variance information. We notice that the rank ordering based on the p-value is also reflected in the average value of the probabilities in the rows. Thus, similar to earlier ranking methods, we could rank order based on the average probability value for any AS, and then choose the top 'r' ASs, subject to the constraint of cost. An extra constraint in choosing the set of ASs could be that every AS chosen should have a return greater than the return of any other AS with a probability greater than 'p' unless the other AS has already been chosen. Formally, this is represented as follows:

Let Chosen Set be represented by **CS**, then

$$AS_i \in CS$$
$$P(AS_i > AS_j) \geq p$$
$$i \neq j$$
$$AS_j \notin CS$$

Substituting the uniform distribution with any other distribution would require the development of a different analytical solution for calculating the probabilities, though the underlying principle will remain the same.

4.6.2 Dealing with Combinations of Strategies Using the Portfolio Theory Framework

The idea of diversification of stocks within a portfolio by investors to reduce the standard deviation on the returns was shown by Markowitz [Markowitz 52]. The central idea behind portfolio theory is to combine two assets in a portfolio that are not perfectly correlated in order to reduce the overall uncertainty on the returns.

The idea behind the portfolio approach to investment is to reduce the overall uncertainty by implementing either uncorrelated or negatively correlated strategies [Butler 99]. For example,

we could have a particular architectural strategy, AS_1, which has high benefit with relatively high cost and uncertainty. Another architectural change, AS_7, could have low benefit and low cost and uncertainty. AS_1 is considered as a high-risk option while AS_7 is a low risk option. Further analysis of the dependencies shows that AS_1 and AS_7 are competing technologies that are mutually exclusive and negatively correlated as far as their uncertainties are concerned. In simpler terms, we find that if AS_1 is successful, AS_7 will not be and vice versa.

This concept leads us to the generalization that there are a number of other changes AS_i and AS_j, which, combined together, could reduce the overall risk of the system even though some of them are doomed to failure. The architectural changes thus chosen could be considered to be a portfolio that attempts to ensure a certain level of success, while simultaneously reducing the overall risk. When combining the various architectural strategies to form a portfolio, one also needs to look at the technical feasibility of implementing all of them at the same time, as well as the implied schedule and budget considerations.

Similar to the problem of rank ordering various ASs, we now consider various ways of combining ASs such that the resultant portfolio maximizes return on investment and minimizes the variance of the portfolio subject to a cost constraint. The return and variance of a portfolio containing ASs is given by:

$$R = \sum_i x_i r_i$$

$$Var = \sum_i \sum_j x_i x_j \sigma_i \sigma_j \rho_{ij}$$

$$where \ x_i = \frac{C_i}{\sum_i C_i}$$

$$and \ \rho_{ii} = 1$$

The return (r_i) is obtained as described in the previous section. The value σ_i is the uncertainty of the return score. This formulation tells us that when we combine strategies that are correlated with each other, then the variance on return for the combination is lower. The correlation value, ρ_{ij}, which lies between +1 and −1, is more challenging to obtain. For this purpose, we introduce the concept of a "dependency structure" between the various ASs under consideration as a proxy for the correlation value. Figure 3 shows the influence of various ASs on the components of a system. The details of the component names are not important for our example.
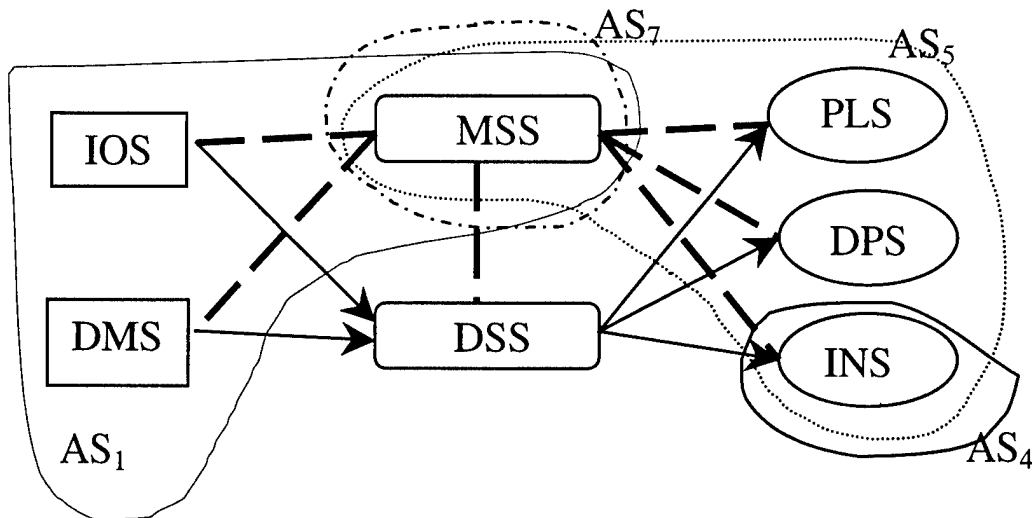
Figure 3: Components and the Influence of ASs

We use this example to show how the dependency structure could be elicited. We see in this example that AS_7 and AS_1 overlap—they affect the same component (MSS). Since they implement competing technologies, they are considered to be uncorrelated (i.e., success of one guarantees failure of the other). Similarly, we see that AS_1 and AS_4 do not have any overlap in terms of components affected. Based on this information, we elicit the correlation information from the experts as shown in Table 4. The information obtained could be qualitative (+/-H, M, L, or 0) or quantitative (a number between −1 and +1).

Table 4: Correlation Matrix for ASs

Arch. Strategy	AS_1	AS_4	AS_5	AS_7
AS_1	1	0	0.3(+L)	-0.6(-H)
AS_4		1	0.6(+H)	0
AS_5			1	-0.3(-L)
AS_7				1

In traditional portfolio management, the investor chooses a particular value of x_i, which is the fraction of total investment in a portfolio in a particular asset. In the case of software, this flexibility is not available since the designers cannot invest in fractions of an AS. However, in a portfolio of ASs, the value of x_i for AS_i varies depending on the chosen set of ASs. If we have 4 strategies in consideration, we can construct 15 portfolios ($^4C_1 + {}^4C_2 + {}^4C_3 + {}^4C_4$) with varying return, cost, and variance. These 15 portfolios are analyzed and a portfolio with a "reasonable" return, cost, and variance is chosen for implementation.[5] We also observe that, as the number of considered strategies increases, the number of possible combinations of portfolios increases in the order n^2 and the calculations become computationally intensive.

---

[5] The project manager determines what is considered "reasonable."

The CBAM, through its idea of quantification of benefit, cost, and uncertainty helps us structure the problem of choosing ASs in the framework of well developed methods from investment theory. The superior techniques now available to us could be applied to the problem for an economically sound solution.

5 Case Study

In this section we describe the application of the CBAM to a real-world software project to demonstrate the feasibility of this method for large-scale projects. The CBAM helped structure an unstructured architecture design problem and offered the project manager a set of solutions to choose from. We first describe the project in brief to give a sense about the magnitude of the project and then outline the various CBAM steps and the results from applying them.

5.1 Description of the Project

The Earth Observing System is a constellation of NASA satellites that gathers data about the Earth for the U. S. Global Change Research Program and other scientific communities worldwide. The Earth Observing System Data Information System (EOSDIS) Core System, also called the ECS, collects data from various satellite downlink stations for further processing. The mission of the ECS is to process the data into higher-form information and make it available in searchable form to scientists around the world. The goal is to provide a common way to store (and hence, process) data, and a public mechanism to introduce new data formats and processing algorithms, thus making the information widely available to the scientific community at large. The ECS processes an input stream of hundreds of gigabytes of raw environment-related data per day. The processing involving the computation of 250 standard "products" results in thousands of gigabytes of information that gets archived at 8 datacenters in the United States. The ECS has important performance and reliability requirements. The long-term nature of the project also makes modifiability an important requirement.

5.2 Applying the CBAM to the ECS

The ECS, with about 1.2 million lines of code in 12,000 modules and about 50 customer off-the-shelf (COTS) products, is a very large software system. We had seven stakeholders, chosen by the project manager, to take part in the CBAM exercise.

5.2.1 Step 1: Choosing the Scenarios and Architectural Strategies (ASs)

Prior to the CBAM, an ATAM exercise was carried out for the ECS project. As a result of this exercise, 72 architectural strategies were identified to improve the system. Except for one strategy that was deemed crucial and non-negotiable, the rest of them needed to be analyzed

with respect to their benefits, costs, and uncertainty. When analyzing an architectural strategy, the decision was whether or not the project should implement that strategy.

5.2.2 Step 2: Assessing the Relative Importance of QAs

To quantify the relative importance of the various QAs, the stakeholders first outlined a list of QAs that they felt were important to the software system's goals. These QAs, as shown in Table 5, were arrived at through consensus and discussion of the definition of each, as well as relating them to the 'utility tree' constructed during the ATAM exercise. The seven stake-holders then independently rated each of the QAs as described in section 4.2. The ratings are shown in Table 6. The stakeholders then discussed the values to arrive at a consensus value to be used through the rest of the method.

Table 5: Description of Quality Attributes (QAs)

Attribute	Description
Maintainability	Lowers maintenance cost of the system by making it easier to find and fix problems and deploy minor changes to existing problems
Operability	To make it cheaper to operate the system or do more load with less people (increase efficiency)
Relavailability	Decreases the degradation in throughput due to downtime; minimizes the operator requirement in recovery of the requests; no key inputs/outputs to the system are lost
Scalability	Stable to increase the problem size, capacity (hardware, operators) scales linearly with workload: system components scale linearly with problem size
Performance	Reduce the end user request latency while increase system throughput
User Satisfaction	Increase user empowerment/capability; decreased response time, accuracy, understandability
Security	The ability to maintain the integrity of their data holding and maintain privacy of the user information and reduce the loss of availability due to denial-of-service attacks
Flextensibility	Ability to insert or add major new functionality or products

Table 6: Quality Attribute Ratings of Stakeholders

Quality Attribute	Stk1	Stk2	Stk3	Stk4	Stk5	Stk6	Stk7	Mean	Var.	Consensus
Maintainability	20	20	20	20	20	15	20	19.286	*3.06*	**19**
Operability	25	25	20	20	20	20	25	22.143	*6.12*	**22**
Relavailability	20	15	15	15	10	15	15	15.0	*7.14*	**15**
Scalability	10	10	5	10	10	10	5	8.571	*5.10*	**9**
Performance	5	10	5	10	15	10	5	8.571	*12.25*	**9**
User Satisfaction	10	15	25	10	15	20	20	16.429	*26.53*	**16**
Security	0	0	5	5	0	0	0	1.429	*5.10*	**1**
Flextensibility	10	5	5	10	10	10	10	8.571	*5.10*	**9**

Using the values elicited in Table 6 we calculate the Kendall's coefficient, W, to be 0.8383, indicating a high level of concordance (F-stat=31.1, p<0.001). This gives us confidence that the stakeholders agreed with each other about the importance of various QAs with respect to the business goals of the system.

5.2.2.1 Pruning the Decision Space

In the triage phase of the method, the stakeholders were provided with the list of 72 architectural strategies that were considered desirable changes to the system. The stakeholders independently rated each AS by QA on a 5-point (-- to ++) scale. The cost estimate for each was also noted on a 3-point scale (H, M, L). The ratings were converted to a quantitative scale (-1 to +1), and a total benefit was computed. The project manager guided the discussion to ensure consensus about the chosen strategies and that no important aspect was being overlooked. Applying a mixed strategy for evaluation, the pruning process resulted in 25 ASs for detailed analysis.

5.2.3 Step 3: Quantifying the Benefits of the ASs

The detailed analysis phase began with the experts independently rating each of 25 ASs by the QAs on a scale of -1 to +1. A template of the rating sheet is shown in Table 7.

Table 7: Template of AS Rating Sheet

QAscore	30	40	15	15	Total	Comment
AS Number	QA1	QA2	QA3	QA4		
AS1						
AS2						
AS3						

The stakeholders varied considerably in their ratings for various ASs and the ranges were recorded as a measure of the uncertainty that the stakeholders had about the effects of the ASs to the system attributes. The concordance score was computed to be 0.326 (F-stat = 2.9, p<0.001), which informs us that it was fairly low, yet still significantly greater than zero. A follow up Delphi exercise was initiated with the experts.

5.2.4 Step 4: Quantifying the Costs of ASs and Incorporating Schedule Implications

The costs of various ASs were assessed in terms of person-months. This estimate was obtained from two of the stakeholders who independently performed a cost estimation exercise.

No guidance regarding cost estimation was given and the experts applied existing methods. The two stakeholders were found to be consistent in their estimates.[6]

5.2.5 Step 5: Calculating the Return for Each AS

The return for the various ASs was calculated using the formula specified in section 4.5. The results are shown in Table 8. The uncertainty in the form of range as well as standard deviation was also calculated.

5.2.6 Step 6: Rank Ordering and Applying an Appropriate Decision-Rule

The ASs were then rank ordered based on their mean and median return score. To determine the sensitivity of this rank ordering, we then calculated the probability matrix as explained in section 4.6.1. The resulting rank ordering of our 25 ASs by the mean, median, and probability is shown in Table 9.

Table 8: Aggregated Benefit Scores, Cost Values, Return Score of Top 10 ASs

AS Number	AGGREGATE BENEFIT SCORE				COST (p. m.)	Return	
	MIN	MAX	MEAN	MEDIAN		MEAN	MEDIAN
AS-2	6	53	33.88	40	4	8.47	10.0
AS-3	13	53	34.47	34	10	3.45	3.4
AS-5	5	51	26.14	25	12	2.18	2.1
AS-6	33	69	52.69	56	6	8.78	9.3
AS-7	30	67	49.29	46.8	6	8.22	7.8
AS-8	17	64	42.58	42.6	16	2.66	2.66
AS-9	36	90	59.48	55.8	20	2.97	2.79
AS-11	24	93	58.74	46.8	16	3.67	2.93
AS-13	24	59	44.71	42.8	16	2.79	2.68
AS-20	45	86	63.37	62.5	16	3.96	3.91

---

[6]   The original estimates differed considerably until a follow-up discussion established that one stakeholder had included cost of testing while the other hadn't. The ensuing adjustment resulted in an almost perfect correlation in cost estimates.

Table 9: Rank Ordering of ASs by Criterion (Top 10)

Arch. Strategy	Return based Rank			Cost based Rank
	Mean	Median	Probab.	
AS-2	2	1	3	1
AS-3	6	5	6	4
AS-5	12	13	11	5
AS-6	1	2	1	2
AS-7	3	3	2	2
AS-8	10	10	9	6
AS-9	8	8	7	10
AS-11	5	7	5	6
AS-13	9	9	8	6
AS-20	4	4	4	6

5.2.6.1 Portfolio Theory Framework to Pick a Set of ASs

In section 4.6.2 we described a technique to incorporate the uncertainty as well as the dependency structure of the various ASs to make a choice. This involved the elicitation of a dependency structure among the ASs, which was used to populate the correlation matrix. This matrix was then used to build portfolios containing various ASs. The return, variance, total benefit, and total cost were computed for each of these portfolios. Considering that we had 25 ASs, the total number of possible combinations is given by

$$^{25}C_1 + {}^{25}C_2 + \ldots + {}^{25}C_{25} = 33,554,431$$

Considering that we have a budget and schedule constraints, we assume that the total number of ASs that can eventually be implemented cannot exceed 8; we restrict the maximum number of ASs in our chosen portfolio to be 8. We also assume that to get a reasonable benefit, the portfolio must contain at least 4 ASs. Even with this restriction, the total number of possible combinations is 1,805,155 ($^{25}C_4 + {}^{25}C_5 + \ldots + {}^{25}C_8$). This is still a fairly large number and the analysis and comparison of portfolios took about 2.5 hours on a Pentium III, 450MHz processor. A portfolio that belongs to a set of portfolios that is non-dominated on the criteria of return and variance is called an efficient portfolio. The efficient portfolios are shown in Figure 4.
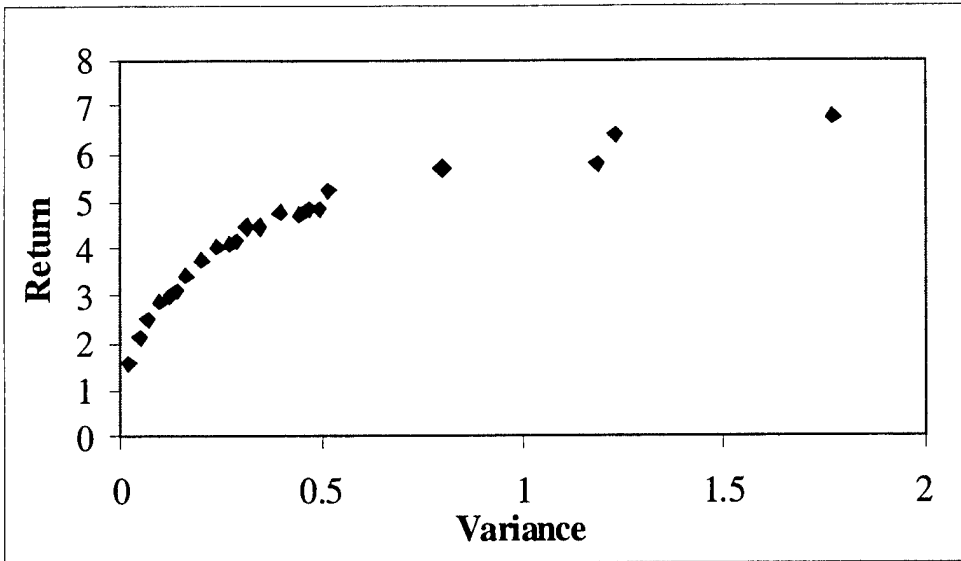
Figure 4: Efficient Portfolios of Architectural Strategies

5.3 Summary of the ECS Case Study

In this section we illustrated how we applied the CBAM to NASA's ECS project. The feedback we received from the ECS project manager indicates that the method holds significant promise for facilitating the decision making process. A benefit perceived by the stakeholders was the structured approach to prune their decision space for changes. The quantification assisted them in objectively assessing the various ASs and implementing those that would give them the greatest return. The method aided the stakeholders in determining how to allocate limited resources to their software evolution effort and they participated enthusiastically.

6 Related Work

Traditional software design methods address the technical aspects of a software system by identifying the functionality to be delivered during the requirement specification phase of product development. Analysis of the product with respect to its economic impact, risks, and tradeoffs is seldom performed. If done, it mainly focuses on the cost of the system and the methods of estimating cost accurately. Identification of the source of benefit for a software system and its quantification is rarely carried out. Some attempts have been made to tie the business goals of the system to the technical aspects of a software system. The Spiral model of software development [Boehm 88] and the Next Generation Process Model (NGPM) [Boehm 94] help system stakeholders converge on a system's next-level objectives, constraints, and alternatives. The NGPM uses the Theory W, which involves identification of the system's stakeholders and their respective "win" conditions. Using a negotiation process among the stakeholders, a mutually satisfactory set of objectives, constraints, and alternatives is determined. Our method differs from the NGPM in the aspects of the stage in the life cycle of the development process, as well as the level of abstraction within the software system. The NGPM addresses the requirements specification stage and tries to understand the broad functional requirements, while our method is at the architectural level of abstraction and expects that the functional requirements are understood and already carried out. In some sense the CBAM is complementary to the NGPM.

There are a few studies on understanding the quality attributes of a software system. Johansson et al. [Johansson 01] use a survey method to understand how different stakeholders view different quality attributes and how they rank the importance of each. They surveyed the software architect, system designer, and marketing manager of three different organizations (one educational, two commercial) as stakeholders. The results from the study indicated that different stakeholders prioritize the various quality attributes differently in spite of a common organizational goal. They also conclude that there are inadequate metrics to measure the impact of the quality attributes of software platforms. This leads to stakeholders not knowing the quality attributes they need to focus on for improvements. An area of further research they have identified is the feedback loop to the stakeholders about how the various quality attributes affect cost, quality, and lead-times for projects built on a software platform. Since our method directly follows an ATAM, the system stakeholders and experts already have a common level of understanding about the system goals and the attributes of importance. In the CBAM we also revisit the definition of the various QAs and make sure that all the stakeholders understand as well as agree about the definition of each and how they affect the system goals.

Earlier in this report, we also mentioned the work of Sullivan et al. [Sullivan 99] and Butler et al. [Butler 99] in the area of application of advanced economic theories to software design. Benaroch and Kauffman [Benaroch 99], in their application of real-option theory to an information system, analyze the decision from a pure business context and do not in any way tie it to the specifics of the system design or implementation. In the CBAM we incorporate the idea of portfolio theory into the design decisions such that the designers can reason about the implementation of certain strategies. In our method, we expect the designers to think about aspects of ASs, like correlation between ASs, as well as implementation of some ASs in order to increase the information about other strategies. This should result in a reduction of variability of the overall outcome.

7 Lessons Learned and Further Developments of the CBAM

In our pilot implementation of the method with NASA's ECS project we gained significant insights into the practical operational issues of running a CBAM. Significantly, we noticed that there were large variations in the elicited values of the contribution scores for the ASs. We feel that this was due to the fact that the quality attributes (performance, modifiability, etc.) for the stakeholders and experts are abstract entities and hence hard to interpret consistently. Also, quantifying the expected utility level of an AS without understanding of the current system QA response/utility level makes the elicitation exercise prone to variable interpretations and judgments amongst the stakeholders. For these reasons we are already planning further refinements to the CBAM.

To quantify and understand the impact of the ASs on the QAs in a consistent manner the stakeholders need a more concrete representation of the QAs. For this purpose we intend to make use of the scenarios as elicited during the ATAM for characterizing the specifics of the various QAs. These scenarios also specify the stimulus-response characteristics of the subsystem in question and provide a concrete example by which the stakeholders can rate the effect of the ASs. The scenarios also provide a basis upon which the system utility is derived and hence it is also possible to rate their relative importance.

One final important feature of the next version of the CBAM that we are developing is that of multiple iterations, where each iteration adds some information and pares down the space of scenarios considered. For example, separate iterations will consider the side effects of ASs, and the correlation between ASs.

8 Conclusion

When developing or evolving any engineering artifact, including software-intensive systems, one is always faced with the problem of determining where to spend a finite set of resources. The ever-present (albeit sometimes implicit) problem is to maximize the benefits resulting from developing or evolving the system given a limited pot of dollars to spend. Frequently this involves selecting the right set of product features. Determining the "right" feature set is clearly necessary for a system's eventual success, but it is not always clear that this is not sufficient. The quality attributes associated with each feature are key. A feature with poor performance or poor security may be equivalent to or worse than not having the feature at all. Moreover, it is the quality attributes (such as performance, security, reliability, modifiability, etc.) that determine the overall architecture of a large and complex system. Usually a significant amount of effort is dedicated to getting these attributes right. Consequently, we have developed and prototyped a decision-making method that explicitly accounts for the cost and benefits associated with engineering qualities into the software architecture of a system. In the Cost Benefit Analysis Method (CBAM), costs and benefits are qualities that are traded off, in addition to the technical quality attributes.

In this report we have introduced the CBAM. The CBAM is

- a method that causes the stakeholders to quantify the benefits as well as the costs, dependencies, and schedule implications of architectural decisions
- a method that explicitly captures the uncertainty surrounding costs and benefits
- a scalable method; it can be inexpensively employed for triage or it can be used exhaustively once the search space of ASs has been narrowed

We have presented the steps of the CBAM, the theory behind the steps and the assumptions we make to suit the theory to our practical needs. We applied the method to NASA's ECS project and our experience has helped us understand the shortcomings and areas of the method that need improvement. Feedback elicited from the stakeholders of the ECS project highlighted the following positive aspects of the CBAM:

- its structured approach to prune a large decision space of ASs
- its objective methodology to allocate limited resources to a software evolution effort
- its ability to encourage a structured discussion regarding the desired properties of the system as well as the course of action for implementation

An identified drawback of the CBAM is that it depends on the subjective judgment of the stakeholders. Formal verification is either very expensive or difficult to carry out. Due to the subjective nature of stakeholder judgment, the elicited values can show large variability. While some of this variability reflects the underlying uncertainty, the rest is due to limited knowledge among the stakeholders regarding all parts of the system. Within the CBAM the Delphi approach is used to eliminate the variability arising due to limited knowledge. This involves stakeholders sharing information about the rationale used while rating particular strategies. This should help make assumptions explicit and more open to scrutiny.

We are also examining the CBAM for places where our approach might be strengthened. We have presently identified three areas of future work:

1. examining how to better elicit, represent, and understand the method's sensitivity to uncertainty

2. exploring how real-option theory can be used to exploit the "wait and watch" nature of many decisions

3. characterizing and using dependencies amongst ASs in a "portfolio of architectural options"

In conclusion, we are optimistic about the prospects of the CBAM evolving to be a theoretically sound and yet practical method that can be used for making software-related cost-benefit decisions.

References

[Benaroch 99]   Benaroch, M. & Kauffman, R.J. "A Case for Using Real Options Pricing Analysis to Evaluate Information Technology Project Investments." *Information Systems Research 10*, 1 (March 1999), 70-86.

[Boehm 81]   Boehm, B.W. *Software Engineering Economics*. Englewood Cliffs, New Jersey: Prentice-Hall, 1981.

[Boehm 88]   Boehm, B.W. "A Spiral Model of Software Development and Enhancement," *Computer 21*, 5 (May 1988), pp.61-72.

[Boehm 94]   Boehm, B.; Bose, P.; Horowitz E.; & Lee, M.J. "Software Requirements as Negotiated Win Conditions." 74-83, *Proceedings of the First International Conference on Requirements Engineering (ICRE94)*, April 18-22, 1994, Colorado Springs, Co. Los Alamitos, Ca.: IEEE Computer Society Press, 1994.

[Brealey 81]   Brealey, R. & Myers, S. *Principles of Corporate Finance*. New York: McGraw-Hill, 1981.

[Butler 99]   Butler, S.; Chalasani, S.; Jha, S.; Raz, O.; & Shaw, M. "The Potential of Portfolio Analysis in Guiding Software Decisions." *First Workshop on Economics-Driven Software Engineering Research*, 1999. <http://www.cs.virginia.edu/~sullivan/EDSER1/>.

[Dixit 94]   Dixit A. & Pindyck, R.S. *Investment Under Uncertainty*. Princeton, New Jersey: Princeton University Press, 1994.

[Fleiss 81]   Fleiss, J.L. *Statistical Methods for Rates and Proportions*. New York: John Wiley & Sons, 1981.

[Johansson 01]   Johansson, E.; Wesslen, A.; Brathall, L.; & Host, M. "The Importance of Quality Requirements in Software Platform Development–A Survey." *Proceedings of the 34th Hawaii International Conference on System Sciences,* Maui, Hawaii, January 3-6, 2001. Los Alamitos, Ca.: IEEE Computer Society Press, 2001.

[Jones 99]   Capers Jones, T. *Estimating Software Costs*. New York: McGraw-Hill, 1999.

[Kazman 00]   Kazman, R.; Klein M.; & Clements, P. *ATAM: A Method for Architecture Evaluation* (CMU/SEI-2000-TR-004, ADA382629). Pittsburgh, Pa.: Software Engineering Institute, Carnegie Mellon University, 2000. <http://www.sei.cmu.edu/publications/documents/00.reports/00tr004.html>

[Keeney 76]   Keeney, R.L. & Raiffa, H. *Decisions with Multiple Objectives: Preferences and Value Tradeoffs*. New York,: John Wiley & Sons, 1976.

[Kendall 55]   Kendall, M.G. *Rank Correlation Methods*. London: C. Griffin, 1955.

[Linstone 75]   Linstone, H. & Turoff, M. *The Delphi Method: Techniques and Applications*. Reading, Ma.: Addison-Wesley, 1975.

[Markowitz 52]   Markowitz, H.M. "Portfolio Selection." *Journal of Finance 7* (March 1952), 77-91.

[Morgan 90]   Morgan G. & Henrion, M. *Uncertainty: A Guide to Dealing with Uncertainty in Quantitative Risk and Policy Analysis*. New York: Cambridge University Press, 1990.

[Shaw 96]   Shaw, M. and Garlan, D. *Software Architecture: Perspectives on an Emerging Discipline*. Upper Saddle River, New Jersey: Prentice-Hall, 1996.

[Sullivan 99]   Sullivan, K.J.; Chalasani, P.S.; Jha, S.; & Sazawal, V. "Software Design as an Investment Activity: A Real Options Perspective." *Real Options and Business Strategy: Applications to Decision Making*, L. Trigeorgis, ed. London: Risk Books, 1999.

[Winterfeldt 86]   Winterfeldt D. & Edwards, E. *Decision Analysis and Behavioral Research*. New York: Cambridge University Press, 1986.

Appendix A: Multi-Attribute Decision Theory for a Software Design Problem

The multi-attribute problem of software design can be stated as follows: Let a be a feasible alternative in the set of architectural strategies (ASs), A. Each alternative 'a' is associated by indices of value $X_1(a)$, ... $X_n(a)$. We can think of these n indices X_1, ... X_n as evaluators of 'a' mapping onto a consequence space. These evaluators are the quality attributes (QAs) of a software system, which we have described earlier. The consequence space is the resultant quality attribute level for the system due to the prescribed AS, a. If $(x_1,..., x_n)$ is a point in consequence space that maps a, then we can never compare x_i and x_j where $i \neq j$. This would be equivalent to comparing performance and maintainability. Hence the software designer's objective is to choose a in A such that he or she is happiest with the payoff of $\{X_1(a),$ ... $X_n(a)\}$. To compare different alternatives we need to define a *value function*, v, which combines $X_1(a)$, ..., $X_n(a)$ into a scalar index of preferability. The value function must be defined on the consequence space with the following property:

$$v(x_1, x_2, ... x_n) \geq v(x_1', x_2', ... x_n') \Leftrightarrow (x_1, x_2, ... x_n) \geq (x_1', x_2', ... x_n')$$

This implies that if we 'prefer' the point $(x_1, x_2, ...x_n)$ over the point $(x_1', x_2', ...x_n')$ in the consequence space, then the value of $(x_1, x_2, ...x_n)$ should be greater than the value of $(x_1', x_2', ...x_n')$. The value function is referred to by many names in literature – ordinal utility function, preference function, worth function or utility function. Given v, the software designer's problem is to choose a in A such that $v(a)$ for a given cost is maximized. The value function v serves to compare the various levels of the different attributes $(X_1, X_2, ... X_n)$ on a similar scale.

For any architectural strategy, $a \in A$, there is a consequence on the quality attributes, $x \in R$. The set of consequences of R that are not dominated is called the *efficient frontier*. Figure 5 depicts a consequence space for the case of two attributes (n=2). Each circle is the consequence of an architectural strategy and all the dark circles show the efficient frontier. The value function v will help the software designer make a decision as to which of the points on the efficient frontier is superior. The case of two attributes is simple and can be plotted easily, however, in practice we usually have more than three quality attributes to care about and that cannot be graphically depicted.
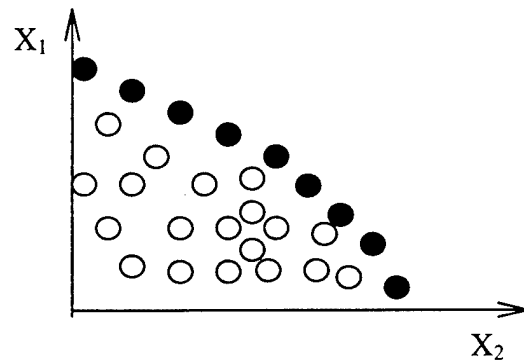
Figure 5: Maximized Boundary for a Two-Attribute Problem

Before we step into the details of value functions, let us examine other ways that the consequence space can be limited. While building software systems, rarely do we maximize a single quality attribute and not care about how low the other attributes are set as a consequence. For example, there are situations where the system is completely useless if the availability is less than 99.9% or the performance is worse than 10 transactions/sec. Hence we can define a set of *threshold levels*, $x_1^o, x_2^o, \ldots, x_n^o$, for each attribute respectively. These threshold levels set the minimum values that any attribute is permitted to be and thus acts as a constraint on the original set of the consequence space as shown in Figure 6. The dotted circles represent the architectural strategies that are not in the feasible space and the dark circles represent the new efficient frontier.
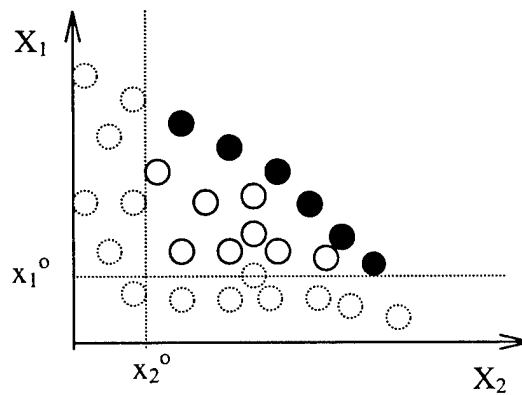

Figure 6: Maximized Boundary with Thresholds

For our problem of assessing architectural strategies and the impact on the software system, we assume an additive preference structure for the quality attributes. Formally this could be expressed as

$$v(\vec{x}) = v_{x_1}(x_1) + v_{x_2}(x_2) + \ldots + v_{x_n}(x_n) = \sum_j v_{x_j}(x_j)$$

This preference structure assumes that the attributes of the system are preferentially independent of one another. A pair of attributes X and Y are said to be preferentially independent of a third attribute Z if the conditional preferences in the (x, y) space do not depend on z. For example, if we consider the quality attributes maintainability, operability, and performance, then we assume that the relative importance of operability with respect to maintainability is independent of the performance level of the system. We believe that this assumption holds true for a majority of the ranges of the quality attributes within which the architectural changes in question affect the system.

To make this form operational we use an additive value function that is scaled to reflect the relative importance of the various attributes with respect to the business goals of the system. The form of the equation is

$$v(\vec{x}) = \sum_i \lambda_j v_j(x_j) \text{ ; where}$$

$$\sum_i \lambda_j = 1$$

$$\forall j : \lambda_j \geq 0$$

We thus need to determine appropriate values for λ (scaling parameters) and $v_i(\ )$.

Ideally, we would like to obtain the value function, v, from existing prototype models that will give us quantitative and fairly accurate insight into the effect of any particular AS on the QAs of the system. However, in particularly large and complex systems, building of prototypes may not always be feasible. In these cases we elicit the expected behavior of the system from various stakeholders such as the main software architect, implementers of various modules, maintainers of various subsystems and others that are interacting with the users on a regular basis. The choice of the stakeholders is made by the project manager(s). The elicitation exercise is divided into two parts:

1. determining the scaling parameters λ_j, (QAScore$_j$)
2. determining the attribute specific utility values $v_j(x_j)$, (ContribScore$_{i,j}$)

Once the various quality attributes of importance are determined, the scaling parameter QAscore$_j$ for each quality attribute, j, is elicited. By design, we ensure that

$$\sum_j QAScore_j = 100$$

$$\text{and } \forall j : QAScore_j \geq 0$$

The value function, $v(\ )$, is typically elicited in units of currency, such as dollars. To account for the risk-averseness of project managers as well as the difficulty in quantifying values in

dollars, we use utility estimates for the various ASs. In this section we elicit the utility values u[$v(\mathbf{x})$] for each of the architectural strategies.[7] A single architectural strategy affects more than one QA. Some effects are positive, some negative, and all to varying degrees. When considering large software systems, there is uncertainty regarding the exact effect of a particular architectural strategy on the system. While the effect of some changes could be measured approximately through simulation models, the effect of some changes can only be obtained through expert elicitation of the experts of the system and their understanding of system behavior. Another source of uncertainty and subjectivity is the utility assessment of various levels of the quality attributes. The sources of uncertainty are depicted in Figure 7.
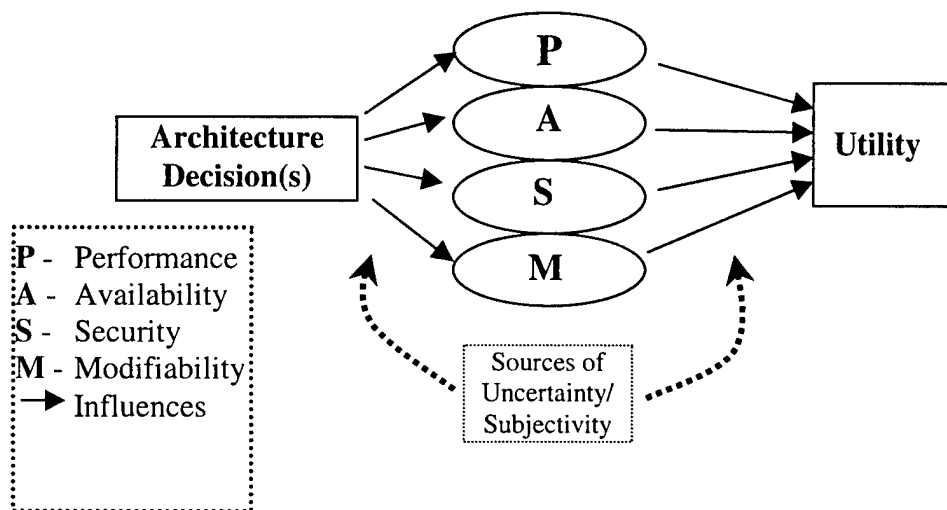

Figure 7: Sources of Uncertainty in Architectural Benefit Assessment

Constructing the utility function, u[$v(\mathbf{x})$], is context specific, hence we tackle this problem by eliciting the utility values of u[$v(\mathbf{x})$]≡u[$\mathbf{x}$] directly from the experts[8] of the system. We ask the experts to rank each architectural strategy (AS$_i$) in terms of its contribution (ContribScore$_{i,j}$) to each quality attribute (QA$_j$) on a scale of −1 to +1. A plus 1 implies that the AS has a perceived best possible effect on the QA for the system and a minus 1 means the opposite. These values represent the utility gained or lost by making a particular architectural change (AS$_i$) on each of the quality attributes (QA$_j$).

During the assessment, we expect the experts to factor into their utility values the short-term as well as long-term effects of that particular change. The benefits and costs accrued in the future are discounted for present values. We assume that the experts in their assessment appropriately discount the future and the net present utility is reflected in their ratings.

---

[7] Here we make a distinction that $v( )$ is the value function with units of dollars and u[$v( )$] is the utility function with units of 'utils.'

[8] We assume that the stakeholders are also the experts of the system.

By asking the experts to assign a utility value ranging from −1 to +1 we make certain assumptions about the utility functions for the quality attributes of the system. In the earlier section we argued about the validity of an additive value function, $v(\ )$. We extend this assumption to the utility function, $u[v(\ )]$, and assume utility independence amongst attributes. This implies that

$$u(\vec{x}) = u_{x_1}(x_1) + u_{x_2}(x_2) + \ldots + u_{x_n}(x_n) = \sum_j u_{x_j}(x_j)$$

In circumstances of high uncertainty, we expect that the system experts are risk-averse and that the elicited ratings reflect a concave utility function.
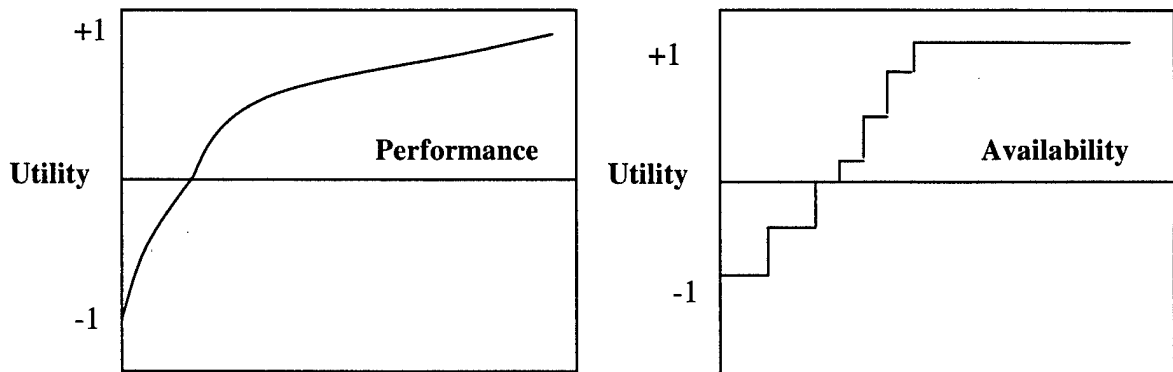

Figure 8: Expected Utility Functions of QAs

Figure 8 depicts what we assume to be examples of the utility function characteristic of the quality attributes of the system. Though these are not actual utility curves, we expect that over time and repetition of the CBAM, the experts of the software system will be able to build similar curves to aid the elicitation process and make it more objective.[9] Ideally, for the contribution scores, we would want to be able to understand the architectural change in terms of the QA response characteristics and then subsequently read off the utility values from a QA-utility chart. That process would involve considerable effort and participation from system experts, and with our present work we are moving in that direction.

---

[9] To generate these curves would also involve an elaborate elicitation procedure, which is not discussed here.

Appendix B: Determining the Probability of Dominance of AS Return Values

Assuming that the underlying distribution is uniform, each architectural strategy is plotted along a single scale, according to its return score. There could thus be two kinds of overlap of scores between any two ASs. The first kind of overlap is when there is a partial overlap of values depicted by Figure 9, while the second kind is when the values of one AS completely overlap another AS, depicted by Figure 10.

8.1 Case 1: Partial Overlap
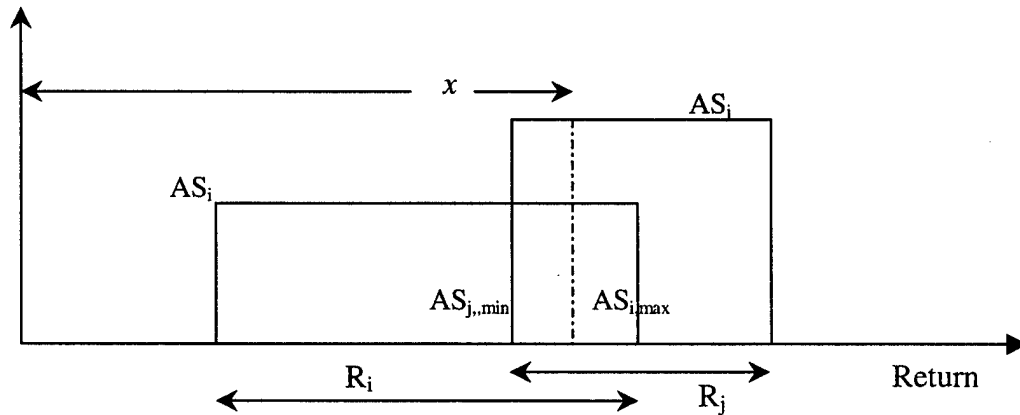
Figure 9: Case 1: Partial Overlap of ASs

Each rectangle represents the probability distribution function of an AS's return score. If R_i is the range of elicited return scores, then the distribution height, h_i, will be determined by normalization:

$$1 = R_i h_i \Rightarrow h_i = \frac{1}{R_i}$$

The probability that the architectural change "i" whose return "value" satisfies $AS_{i,min} \le value \le AS_{i,max}$ is larger than the return "value" of architectural change "j" is

$$prob\left(AS_i \geq AS_j \cap i \neq j\right)$$

$$= prob\left(x \geq AS_j \cap x \leq AS_i \cap i \neq j \; \forall AS_{j,\min} \leq x \leq AS_{i,\max}\right)$$

$$= \frac{1}{R_i R_j} \int\limits_{AS_{j,\min}}^{AS_{i,\max}} dx_j \int\limits_{x_j}^{AS_{i,\max}} dx_i$$

$$= \frac{1}{R_i R_j} \int\limits_{AS_{j,\min}}^{AS_{i,\max}} dx_j \left(AS_{i,\max} - x_j\right)$$

$$= \frac{1}{R_i R_j}\left\{\left(AS_{i,\max}\right)^2 - \left(AS_{i,\max}\right)\left(AS_{j,\min}\right) - \frac{1}{2}AS_{i,\max}{}^2 + \frac{1}{2}\left(AS_{j,\min}\right)^2\right\}$$

$$= \frac{1}{2R_i R_j}\left\{\left(AS_{i,\max}\right)^2 - 2\left(AS_{i,\max}\right)\left(AS_{j,\min}\right) + \left(AS_{j,\min}\right)^2\right\}$$

$$= \frac{1}{2R_i R_j}\left(AS_{i,\max} - AS_{j,\min}\right)^2$$

$$\text{hence,} \quad prob\left(AS_i \geq AS_j\right) = \frac{1}{2R_i R_j}\left\{AS_{i,\max} - AS_{j,\min}\right\}^2$$

8.2 Case 2: Complete Overlap
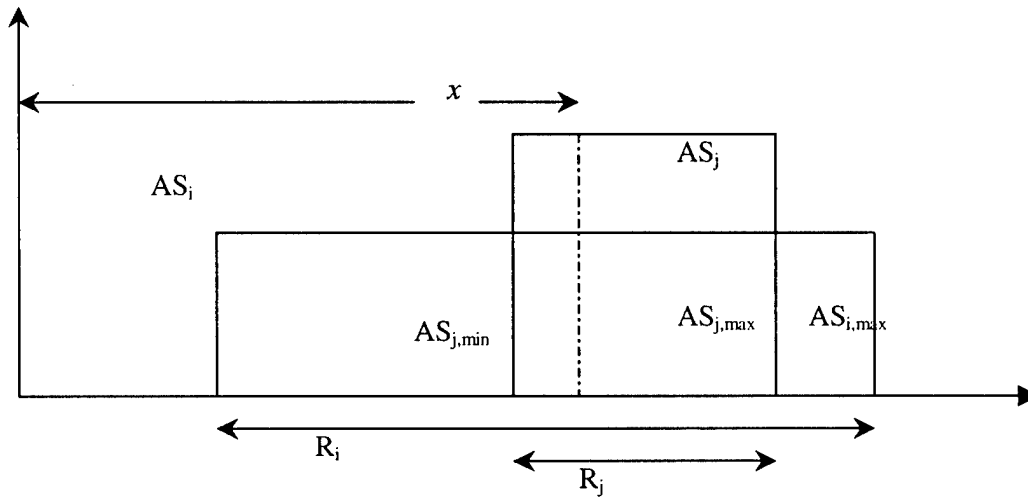
Figure 10: Case 2: Complete Overlap of ASs

$$prob\left(AS_i \geq AS_j \cap i \neq j\right)$$

$$= prob\left(x \geq AS_j \cap x \leq AS_i \cap i \neq j \; \forall AS_{j,min} \leq x \leq AS_{j,max}\right)$$

$$= \frac{1}{R_i R_j} \int_{AS_{j,min}}^{AS_{j,max}} dx_j \int_{x_j}^{AS_{i,max}} dx_i$$

$$= \frac{1}{R_i R_j} \int_{AS_{j,min}}^{AS_{j,max}} dx_j \left(AS_{i,max} - x_j\right)$$

$$= \frac{1}{R_i R_j} \left\{ \left(AS_{i,max}\right)\left(AS_{j,max} - AS_{j,min}\right) - \frac{1}{2}\left(AS_{j,max}^2 - AS_{j,min}^2\right) \right\}$$

$$= \frac{1}{2R_i R_j} \left\{ 2AS_{i,max} R_j - \left(AS_{j,max} + AS_{j,min}\right)R_j \right\}$$

$$= \frac{1}{2R_i} \left(2AS_{i,max} - AS_{j,max} - AS_{j,min}\right)$$

Hence the generic solution is:

Case: 1: When there is partial overlap between AS$_i$ and AS$_j$.

$$prob\left(AS_i \geq AS_j\right) = \frac{1}{2R_i R_j} \left\{ AS_{i,max} - AS_{j,min} \right\}^2$$

Case: 2: When AS$_i$ completely overlaps AS$_j$.

$$prob\left(AS_i \geq AS_j\right) = \frac{1}{2R_i} \left\{ 2AS_{i,max} - AS_{j,max} - AS_{j,min} \right\}$$

REPORT DOCUMENTATION PAGE

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.

1. AGENCY USE ONLY	3. REPORT DATE	3. REPORT TYPE AND DATES COVERED
(Leave Blank)	December 2001	Final

4. TITLE AND SUBTITLE	5. FUNDING NUMBERS
Using Economic Considerations to Choose Among Architecture Design Alternatives	F19628-00-C-0003

6. AUTHOR(S)

Jayatirtha Asundi, Rick Kazman, Mark Klein

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)	8. PERFORMING ORGANIZATION REPORT NUMBER
Software Engineering Institute Carnegie Mellon University Pittsburgh, PA 15213	CMU/SEI-2001-TR-035

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSORING/MONITORING AGENCY REPORT NUMBER
HQ ESC/XPK 5 Eglin Street Hanscom AFB, MA 01731-2116	

11. SUPPLEMENTARY NOTES

12A DISTRIBUTION/AVAILABILITY STATEMENT	12B DISTRIBUTION CODE
Unclassified/Unlimited, DTIC, NTIS	

13. ABSTRACT (MAXIMUM 200 WORDS)

The software architecture forms an essential part of a complex software-intensive system. Architecture design decision-making involves addressing tradeoffs due to the presence of economic constraints. The problem is to develop a process that helps a designer choose amongst architectural options, during both initial design and its subsequent periods of up-grade, while being constrained to finite resources. To address this need for better decision-making, we have developed a method for performing economic modeling of software systems, centered on an analysis of their architecture. We call this method the Cost Benefit Analysis Method (CBAM). The CBAM incorporates the costs and benefits of architectural design decisions and provides an effective means of making such decisions. The CBAM provides a structured integrated assessment of the technical and economic issues and architectural decisions. The CBAM utilizes techniques in decision analysis, optimization, and statistics to help software architects characterize their uncertainty and choose a subset of changes that should be implemented from a larger set of alternatives. We also report on the application of this method to a real world case study.

14. SUBJECT TERMS	15. NUMBER OF PAGES
cost benefit analysis method, portfolio theory, software architecture	56 pp.

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89) Prescribed by ANSI Std. Z39-18 298-102