A Method for Understanding Quality Attributes in Software Architecture Structures

Mikael Svahnberg, Claes Wohlin, Lars Lundberg, Michael Mattsson

Department of Software Engineering and Computer Science
Blekinge Institute of Technology, PO Box 520, S-372 25 Ronneby SWEDEN
Phone: +46 457 385000

[Mikael.Svahnberg|Claes.Wohlin|Lars.Lundberg|Michael.Mattsson]@bth.se

ABSTRACT

To sustain the qualities of a software system during evolution, and to adapt the quality attributes as the requirements evolve, it is necessary to have a clear software architecture that is understood by all developers and to which all changes to the system adheres. This software architecture can be created beforehand, but must also be updated as the domain of the software, and hence the requirements on the software system evolves. Creating an architectural structure for a system or part of a system so that the architecture fulfils the desired quality requirements is often hard. In this paper we propose a decision support method to aid in the understanding of different architecture structure candidates for a software system. We propose a method that is adaptable with respect to both the set of potential architecture structures, and quality attributes relevant for the system's domain to help in this task. The method creates a support framework, using a multi-criteria decision method, supporting comparison of different software architecture structures for a specific software quality attribute and vice versa. Moreover, given a prioritization of quality attributes for the software system, or part thereof, the most suitable software architecture structure can be indicated using the created framework.

Categories and Subject Descriptors

D2.2 [**Design Tools and Principles**] Decision Tables, D2.11 [**Software Architectures**] Patterns.

General Terms

Design, Experimentation

Keywords

Architecture Structures, Quality Attributes, Analytic Hierarchy Process

1. INTRODUCTION

In [16] Parnas describes the phenomenon of software aging. He ascribes this to two causes: (1) the domain changes around the software and (2) changes to the system are introduced in a careless manner, which degrades the system. Part of the solution to both of these problems may be found in having and maintaining a clear and updated software architecture for a software system. As is also described in [16], having an architecture that all changes must be related to will help to prevent the second form of decay. As the domain of the software evolves, so will the requirements on the software system, and hence the architecture needs to be re-evaluated so that it still reflects a modern system that fits the evolved domain. By doing this on a regular basis, we believe that the first form of aging can be, if not hindered, so at least relieved.

Furthermore, an appropriate architecture is not only governed by functional requirements, but to a large extent by quality attributes [2][3][6]. However, knowing this, it is still a non-trivial task to create an appropriate architecture. There are usually more than one quality attribute involved in a system, and the knowledge of the benefits and drawbacks of different architectural structures with respect to different quality attributes is not yet an exact science. Decisions are often taken on intuition, relying on the experience of senior software developers.

This introduces a risk that a particular architectural structure is chosen not based on relevant grounds but because the senior software architect is familiar with aspects of the particular structure and hence favours it over other unknown or less familiar structures. This leads to a situation where the desired quality attributes are designed into the system as an afterthought, which invariably leads to "patchy" and brittle systems that will not withstand the tooth of time.

Quality cannot be added to the system as an afterthought; it has to be built into the system from the beginning. Thus, an architecture structure for any software system ought to be based on functional needs, domain specificities and quality requirements. Hence, support is needed. The focus in this paper is on presenting some key aspects for a method supporting the understanding of architecture structures based on quality attributes. Such a method provides one important input to decision-makers when designing a suitable system structure, together with other considerations.

The term "software architecture structure" is used to denote an architecture, or an architecture proposal of a software system, which can be on any level of granularity, such as a software product, a software module, software subsystem, or software component. In this paper, we only focus on the software artefacts, although the proposed method may be applicable on a system level as well.

1.1 Description of Problem

In this paper, we address the following problem:

> Given a set of requirements on quality attributes for a system and a set of architecture structures, which architecture structure is the most appropriate, i.e. fulfils the quality requirements on the system best?

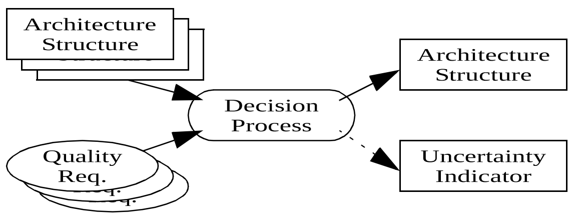

Figure 1. Illustration of Problem

This problem is also illustrated in Figure 1.

If a structured way to understand the benefits and liabilities of different architecture structures is found and used this may increase the confidence in the decisions taken, and hence the software systems developed are not forced into a mould which may not be entirely suitable for the problem at hand. Moreover, by re-evaluating the architecture decisions regularly, it is ensured that the architecture always reflect the current quality requirements, or we are able to strive in the right direction.

A structured way to understand the benefits and liabilities of different architectures is important because, as shown by Johansson et al. [9], different stakeholders tend to have different views of the importance of various quality requirements for a system, and the differing experiences of the software developers may also lead to a different interpretation of the strengths and weaknesses of architecture structures. A structured method facilitates in this situation because it enables us to identify where stakeholders and developers have differing opinions.

Furthermore, as indicated in Figure 1 we are also hoping to be able to generate some form of uncertainty indicator, showing by which degree of certainty the decision process nominates a particular architecture structure.

1.2 Context of Method

The objective of the method proposed in this paper is to enable a quantified understanding of different architecture candidates for a software system. We assume, for the sake of this method, that a small set of architecture structures is developed, after which the proposed method provides support for deciding which of the architecture structures is best suited to meet the quality requirements of the software system. This architecture may either be the architecture that the system is designed according to, or an architecture to strive towards when evolving the system. As is indicated in e.g. [10], architectures degrade over time unless something is made to prevent this. One way may be to have a clear architecture to relate changes to.

It is not our intention that the proposed method should be used as the only method when designing a software architecture structure. Rather, we suggest that the method is used in conjunction with other methods, possibly more qualitative in nature, or focussing on other aspects of the software development, such as the functional requirements. Examples of such methods are the design method proposed by Hofmeister et al. [6], the Rational Unified Process [8] and the QASAR design method [3]. The way we currently see it, these methods are used to create architecture structures, which constitute candidate designs for a system, after which the method proposed in this paper is used to further understand and discern between the strengths and weaknesses of the different architecture candidates, and possibly comparing with the current architecture of the system.

The context of the method is thus a specific situation within a particular company, where either the initial architecture of a system is designed, or the current architecture of a system is evaluated against newer architecture suggestions. The architecture structures are identified within this context, and they are also evaluated within this context. Similarly, the quality attributes are only the ones that are relevant for the domain in question and for the business model of the company.

The purpose of having a clear vision of the architecture is, as outlined earlier, to prevent the software system from aging prematurely. The architecture need thus not ever be actually implemented, but can be an ideal architecture that only defines a goal to aspire for.

1.3 Outline of Paper

The remainder of this paper is organized as follows. In Section 2 we present the proposed method for understanding architecture structures based on quality attributes. In Section 3 we present an example of how to use the proposed method. The method and potential extensions are discussed in Section 4. Finally, the paper is summarized and future work is discussed in Section 5.

2. METHOD

Methods have been developed to evaluate the quality attributes of an architecture [13], and some extensions to incorporate costs have recently been presented [14], but to the best of our knowledge little research has been conducted with respect to quantifiably analysing software architecture structures based on quality objectives.

The objective here is to propose a method for finding a suitable software architecture structure based on the quality requirements on the software product in question. The method is broken down into a number of concrete steps.

1. Identify potential software architecture structures and key quality attributes.
2. Create method framework.
3. Prioritize quality attributes for the software system to be developed.
4. Identify software architecture structure.
5. Determine the uncertainty in the identification.

These steps are illustrated in Figure 2. As can be seen, the first step is performed before the actual analysis process begins, but we include it nevertheless as it is a vital part of any architecture design process to identify potential candidates. Moreover, the FQA, FVC and PQA referred to are outcomes of step 2 and 3, and are further discussed in Section 2.2 and Section 2.3. Each of the steps in Figure 2 corresponds to a subsection below, where the steps are described in further detail.

During the remainder of this paper, we will use a number of acronyms to refer to different sets of data. These acronyms are presented in Table 1.

2.1 Step 1: Identify Candidates

This step is fairly simple to describe, although not trivial in a development situation. The intention is that possible software architecture structures are created, listed and described so that people understand the differences and similarities between them. Further, the key quality attributes for the system to be developed are identified, listed and described.

It is outside the scope of this paper to describe how the architecture structures are created, but as mentioned earlier, various design methods (e.g. [3][6][8]) can be used to create the architecture structures methods, and standard requirements engineering methods (e.g. [5][15]) can be used to obtain the quality attributes.

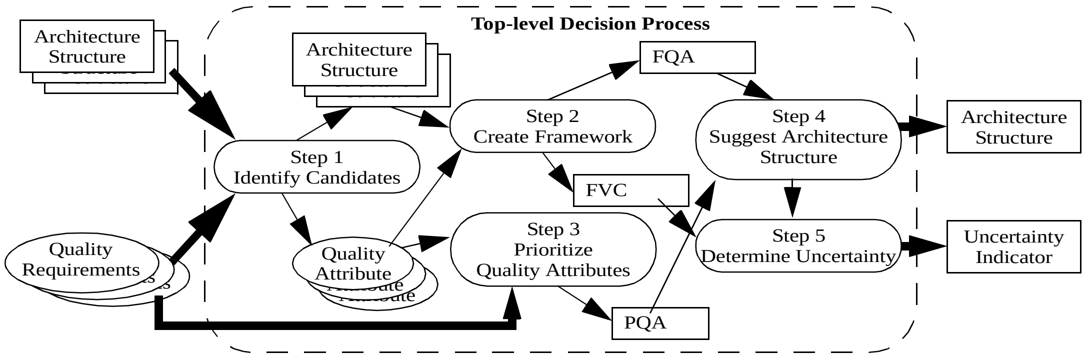

Figure 2. Illustration of Solution

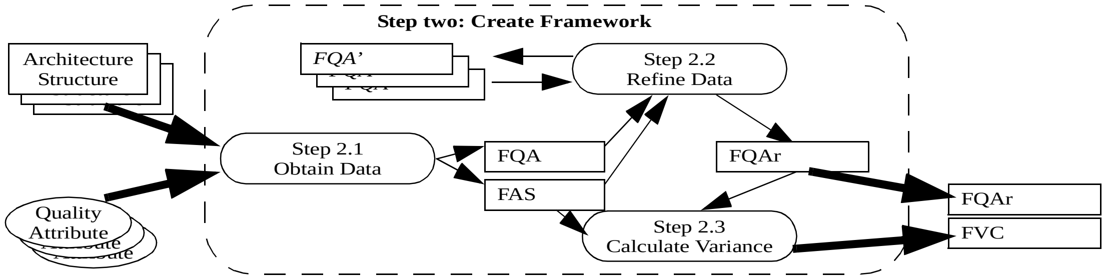

Figure 3. Illustration of step two of solution

The outcome of this step is two lists containing the relevant software architecture structures and quality attributes respectively. As mentioned in Section 1.2, the actual structures and attributes on the lists are dependent on the application domain and the situation in which the software is developed.

We would again like to stress the fact that the architecture structures used can be on any level of granularity on any level of the system (e.g. product, module, subsystem or component). Similarly, we do not put any constraints on the level of granularity for the quality attributes. However, in our experience it is beneficial if all quality attributes are on the same level of granularity, as this facilitates comparison between the quality attributes. Moreover, it may be easier if, as in the example provided in Section 3, the specific quality attributes are grouped into categories to facilitate the prioritization process. The reason why this is easier is simply that the number of inputs to the creation of the framework decreases, which reduces the number of comparisons that need to be made.

Table 1. Acronyms used in paper

Acronym	Description
FQA	Framework for Quality Attributes. A set of vectors where architecture structures are ranked according to their ability to meet particular quality attributes.
FAS	Framework for Architecture Structures. A set of vectors where the support for different quality attributes are ranked for each architecture structure.
FVC	Framework for Variance Calculation. A vector of variance indicators for a FQA vector set.
PQA	Prioritized list of Quality Attributes. A list of quality attributes prioritized for a system to design.
FQA'	An intermediate result during the process of generating the FQAr.
FQAr	A refined version of the FQA, where the values of FAS have been used to increase the accuracy.

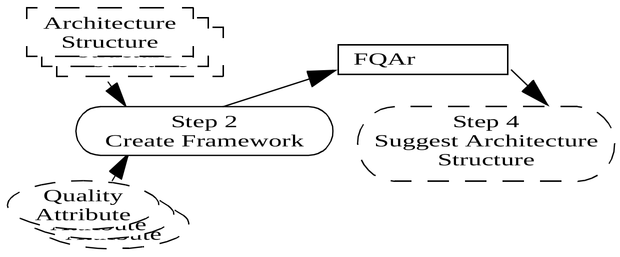

Figure 4. Refinements to Process

This step produces the inputs to the decision process in Figure 1, in that it generates a set of *n* architectural structures and a set of *k* quality attributes.

2.2 Step 2: Create Framework

In this section we go through step two of the proposed method in further detail. In Section 2.2.1 we discuss how the data is obtained that is used to create the framework. In Section 2.2.2 we discuss how this data, formed into two tables with a set of vectors in each (we call these vector sets *Framework for Quality Attributes (FQA)* and *Framework for Architecture Structures (FAS)*, respectively), are related to each other and how the FAS can be used to improve the accuracy of FQA. As a result from this step a refined vector set (called the *FQAr*) is generated. A final outcome of correlating FQA and FAS is that we get an indication of the variance in the data (represented in the vector *FVC*). How this is obtained is presented in Section 2.2.3. These sub-steps within step two, and the different vector sets created during the process are illustrated in Figure 3. Putting this into the context of the rest of the method, there is a minor change to the solution outlined in Figure 2, and these changes are illustrated in Figure 4 (for clarity, we have left out processes and data sets not involved in the refined view).

Table 2. Example FQA - Row normalized vector set

	AS 1	AS 2	AS 3	AS 4	Sum
QA 1	$FQA_{1,1}$	$FQA_{1,2}$	$FQA_{1,3}$	$FQA_{1,4}$	1
QA 2	$FQA_{2,1}$	$FQA_{2,2}$	$FQA_{2,3}$	$FQA_{2,4}$	1
QA 3	$FQA_{3,1}$	$FQA_{3,2}$	$FQA_{3,3}$	$FQA_{3,4}$	1
QA 4	$FQA_{4,1}$	$FQA_{4,2}$	$FQA_{4,3}$	$FQA_{4,4}$	1

2.2.1 Obtaining Data for Framework

The basic idea of our method is that it is possible to understand how good certain software architecture structures are for different quality attributes. This implies that we can determine two things:

- A comparison of different software architecture structures for a specific software quality attribute.
- A comparison of different quality attributes for a specific software architecture structure.

To succeed in this a method for ranking software architecture structures and quality attributes respectively is needed. Such methods are available from the management science literature, for example in Anderson et al. 2000 [1]. The methods are often denoted multi-criteria decision processes. One such method is the Analytic Hierarchy Process, which was originally proposed by Saaty 1980 [17]. This approach has been applied in software engineering by other researchers addressing, for example, requirements engineering [11] and project estimation [18]. The Analytic Hierarchy Process can be used to prioritize different items or aspects. The result is a priority vector with relative weights on the different items or aspects being prioritized. A study has been undertaken to illustrate how the method framework can be determined when having determined a suitable set of architecture structures and quality attributes [19].

Applying such a method as the Analytic Hierarchy Process in our context means that two sets of vectors may be created. The first set of vectors is connected to prioritizing the software architecture structures with respect to a certain quality attribute. This means that it is possible to determine the order in which different software architecture structures are believed to support a specific quality attribute. Moreover, it is possible to get a relative weight using the Analytic Hierarchy Process. The latter means that it is possible to determine how much better a specific structure is with respect to a quality attribute. The outcome of such a comparison is discussed in Section 3.

It should be noted that the AHP method, by using a set of n*(n-1)/2 pair-wise comparisons, creates a vector where all the alternatives are prioritized and relative weights are assigned to each alternative. This vector is normalized such that all values are between 0 and 1, and together sum up to 1. As the method uses pair-wise comparisons and, ultimately, subjective measurements, creating this vector is a fairly simple task. This method has successfully been used in e.g. Karlsson and Ryan 1997 [11] and Karlsson et al. 1998 [12].

In this step of our proposed method, we use the AHP method to create vectors signifying the relative support for different quality attributes within architecture structures (FAS) and the relative ranking of how well different architecture structures support different quality attributes (FQA).

For this paper, it is not relevant to describe in detail how the set of vectors is created. Instead, we present how such a set may look in Table 2. In the table, we see four different architecture structures, numbered from one to four, and four different quality attributes, equally numbered from one to four. For each of the quality attributes, the architecture structures relative support for the quality attribute in question is shown. The values, denoted $FQA_{i,j}$ in this example, are normalized so that each row sums up to 1.

Table 3. Example FAS - Column normalized vector set

	AS 1	AS 2	AS 3	AS 4
QA 1	$FAS_{1,1}$	$FAS_{1,2}$	$FAS_{1,3}$	$FAS_{1,4}$
QA 2	$FAS_{2,1}$	$FAS_{2,2}$	$FAS_{2,3}$	$FAS_{2,4}$
QA 3	$FAS_{3,1}$	$FAS_{3,2}$	$FAS_{3,3}$	$FAS_{3,4}$
QA 4	$FAS_{4,1}$	$FAS_{4,2}$	$FAS_{4,3}$	$FAS_{4,4}$
Sum	1	1	1	1

Table 4. FQA vector set

	AS 1	AS 2
QA 1	0.6	0.4
QA 2	0.3	0.7

Table 5. FAS vector set

	AS 1	AS 2
QA 1	0.5	0.6
QA 2	0.5	0.4

The second set of vectors is obtained in the same way, but here the different quality attributes are prioritized for a specific software architecture structure. This means that the relative support for a quality attribute of a structure can be determined. This set of vectors are shown in Table 3 with values $FAS_{i,j}$ in the cells. It should be noted that, for example, both $FQA_{1,1}$ and $FAS_{1,1}$ are measures of support for QA 1 by SA 1, although from different perspectives. This fact is used as part of the method to determine the uncertainty in the identification of an appropriate software architecture structure. This is further elaborated in Section 2.2.2 and Section 2.2.3.

These two sets of vectors provide a framework for working with software architecture structures with respect to software quality attributes. The main objective of the current work is to use the framework to indicate a suitable software architecture structure for a system to design.

2.2.2 Adjusting the FQA

The fact that we are able to do both row-wise comparisons (FQA) and column-wise comparisons (FAS) opens up some possibilities to increase the quality of our estimations and at the same time determining the uncertainty in our predictions. The nature of these new possibilities can be understood by considering the following very small example.

Consider a situation with two architectural structures, i.e., $n = 2$, and two quality attributes, i.e. $k = 2$, and assume that we have obtained the FQA and FAS in Table 4 and Table 5.

Let $s_{1,1}$ denote the support provided by Structure 1 for QA 1 and $s_{2,1}$ denote the support provided by Structure 1 for QA 2. Furthermore, let $s_{1,2}$ denote the support provided by Structure 2 for QA 1 and $s_{2,2}$ denote the support provided by Structure 2 for QA 2. From column one in the FAS we see that $s_{1,1} = s_{2,1}$. Furthermore from the FQA we get the relations: $s_{1,1} = 3s_{1,2}/2$ and $s_{2,1} = 3s_{2,2}/7$. From these three equations we get: $7s_{1,2} = 2s_{2,2}$. This is, however, inconsistent with the relation obtained by looking at column two in the FAS, i.e., the relation obtained by this column is $s_{1,2} = 3s_{2,2}/2$. Consequently, the two vector sets are incompatible, which of course is possible (and also highly likely) since the values in the vector sets are obtained from (expert) opinions and experiences.

The fact that the information in the FQA and FAS is incompatible can be used in (at least) two ways: first we can use the information in the FAS for adjusting the values in the FQA, which is our main concern; second, we can calculate a variance value (or some other

Table 6. FQA' based on row one

	AS 1	AS 2
QA 1	0.6	0.4
QA 2	0.6	0.27

Table 7. FQA' based on row two

	AS 1	AS 2
QA 1	0.3	1.05
QA 2	0.3	0.7

Table 8. Normalized FQA' based on row one

	AS 1	AS 2	Sum
QA 1	0.6	0.4	1
QA 2	0.69	0.31	1

Table 9. Normalized FQA' based on row two

	AS 1	AS 2	Sum
QA 1	0.22	0.78	1
QA 2	0.3	0.7	1

Table 10. FQAr

	AS 1	AS 2	Sum
QA 1	0.5	0.5	1
QA 2	0.4	0.6	1

uncertainty indicator) based on the degree of inconsistency between the two vector sets. In this section we discuss some simple techniques for doing this.

The calculation in this section is based on the assumption that the values in the FQA and FAS are of equal quality. Using the AHP method, this can be ascertained as AHP generates a consistency index rating how consistent the answers are with each other. The basic strategy is then to calculate *k FQA'*-vector sets, such that each FQA' is compatible with the FAS and one row in the FQA. Having obtained these *k* vector sets, we adjust the FQA by simply taking the average of these *k* FQA'-vector sets and *k* copies of the FQA thus obtaining the FQAr which we can then use for identifying an appropriate architectural structure (see Section 2.4). The FQA' vector sets stem from the FAS and by taking the average between the *k* FQA'-vector sets and *k* copies of the FQA, the FQA and FAS will contribute equally to the FQAr.

From the discussion above we see that based on the FAS and row one in the FQA it is possible to calculate another row (*i*) in the new FQA'-vector set in the following way:

$$FQA'_{i,j} = \frac{FQA_{1,j}FAS_{i,j}}{FAS_{1,j}}.$$

In order not to complicate the presentation we assume that all involved values are larger than zero.

Similarly, based on the FAS and row two in the FQA it is possible to calculate another row (*i*) in the new FQA'-vector set in the following way: $FQA'_{i,j} = \frac{FQA_{2,j}FAS_{i,j}}{FAS_{2,j}}$.

If we consider the small 2x2-vector sets in Table 4 and Table 5, we get the two FQA' vector sets in Table 6 and Table 7.

We see that the row sums in the two FQA' vector sets are no longer normalized to one. It is not obvious how one should handle this, but we suggest that we add an additional step where the sum of each row is normalized to one, so that the values in the FQA' tables are of the same magnitude as the already normalized FQA table. The normalized FQA' vector sets will thus be as presented in Table 8 and Table 9.

By taking the average of these two vector sets and two copies of the FQA we get the FQAr in Table 10 which was our goal.

Table 11. FVC

	AS 1	AS 2
QA 1	0.036	0.036
QA 2	0.038	0.038

Table 12. Prioritized quality attributes for system to develop.

Attribute	Priority
QA 1	PQA_1
QA 2	PQA_2
QA 3	PQA_3
QA 4	PQA_4
Sum	1

2.2.3 Variance Calculation

Since each value in the FQAr is the average of 2*k* values (*k* times the value in the FQA and the value in the *k* normalized FQA'-vector sets) we can calculate the variance in the ordinary way. We will thus obtain a variance vector set - the *FVC*-vector set. For the small example above we would get the FVC in Table 11.

2.3 Step 3: Prioritize Quality Attributes

The next step of the method is to conduct a prioritization of the quality attributes for the software system in question. Different methods may be applied for prioritization [12]. This includes subjective judgement with or without consensus building and methods such as providing a total sum of points to be divided between the items or aspects you would like to Prioritize. Most methods have however weaknesses and it is mostly hard to judge the goodness of the Prioritization.

The Analytic Hierarchy Process (AHP) addresses some of these problems [17], since it allows for a calculation of a consistency index for the prioritization. This opportunity arises from the fact that AHP is based on all pair wise comparisons of whatever we would like to prioritize. In our example (Table 2 and Table 3), we need to perform six comparisons for the quality attributes, since we have four quality attributes. When using AHP, each pairwise comparison is answered with a number between 1 to 9 in favour of one of the elements in the pair to compare. This enables the creation of tables with numeric values for each element. Hence, the outcome of the prioritization process is a vector (called *PQA*) with relative weights, denoted PQA_i, on the importance of the different quality attributes for the software system in question, see Table 12.

More generally, we have to perform $n \times (n-1)/2$ comparisons, where *n* is the number of items or aspects to prioritize. This is the price to pay when using a method such as AHP: the number of comparisons grows fairly quickly. On the other hand, it is not likely that we have a huge number of architecture structures or quality attributes to prioritize (either for the framework or for the prioritization of quality attributes of the software system to develop). The number of architecture structures is limited by the amount of time and effort it takes to develop them, so in a practical setting there will most likely not be more than a few architecture candidates developed. The number of quality attributes may be larger, but this number can be reduced by grouping the quality attributes into categories, each representing some aspect of the system's requirements. The number of quality attributes can also be reduced by selecting a smaller set of quality attributes to focus on. In our experience, for most software systems there is a small set of

Table 13. Example FAS

	AS 1	AS 2
QA 1	0.4	0.25
QA 2	0.3	0.25
QA 3	0.2	0.25
QA 4	0.1	0.25

Table 14. Example FQA

	AS 1	AS 2
QA 1	0.5	0.5
QA 2	0.38	0.62
QA 3	0.25	0.75
QA 4	0.12	0.88

Table 15. PQA-vector

Attribute	Priority
QA 1	0.4
QA 2	0.3
QA 3	0.2
QA 4	0.1

easily identified quality attributes that are more relevant to consider, and these are the ones that should be used for the framework and the PQA.

Further, it should be noted that each comparison is conducted very quickly. This is based on experience from conducting this type of studies in another area [12]. We have also conducted a related experiment [19], in which we used five architecture structures and six quality attributes. This resulted in 135 questions being asked to each participant to create the framework (the FAS and FQA), which took the participants approximately one hour of concentrated work to complete. (The first result came in after 45 minutes and the last came in after 70 minutes.)

The outcome of this step is a vector with the different quality attributes prioritized, and with a relative weight of their importance. This vector is used in the next step, where the data is used to determine the most appropriate software architecture structure for the given situation.

2.4 Step 4: Suggest Architecture Structure

As discussed above, we can obtain two kinds of vector sets: the FQA and the FAS (plus the derived FQAr). Based on these vector sets and the PQA-vector we are going to identify an appropriate architectural structure. It may at first glance be tempting to try to correlate the PQA-vector with the columns in the FAS, and select the architectural structure which has the most similar profile. Consider for example the following small example with a FAS in Table 13 and a PQA-vector in Table 15.

Without going into details about how we define the correlation, it is clear that the profile of Structure 1 is closer to the PQA-vector than the profile of Structure 2. The obvious conclusion would thus be to select Structure 1 in this case. However, assume that the FAS corresponds to the FQA shown in Table 14.

From the FQA we see that Structures 1 and 2 are equally good for QA 1, and that Structure 2 is better for all other quality attributes, i.e., we should in fact select Structure 2. The example above is somewhat naive since Structure 2 is consistently better than Structure 1. However, the example illustrates that a good correlation between a column in the FAS and the PQA-vector is not a good criterion for identifying an architectural structure. Instead we suggest that the FQA or the derived FQAr should be used for pinpointing a suitable structure.

We therefore suggest a very direct method which is: **Suggest structure i such that** $\sum_{j=1}^{k} PQA_j FQAr_{j,i}$ **is as large as possible.**

2.5 Step 5: Determine Uncertainty

In order to obtain the uncertainty in our suggestion we can calculate the variance for each structure *i* given the PQA-vector and the FVC. From the rules of variance propagation we know that the variance for structure *i* is obtained in the following way $\sum_{j=1}^{k} PQA_j^2 FVC_{j,i}$.

We are thus able to determine the uncertainty in our suggestion.

2.6 Summary

In Section 2.1 architecture structures are created and quality attributes are identified, upon which the FQA, FAS and the FQAr vector sets created in Section 2.2 are based. These are then used to suggest an architecture structure in Section 2.4.

Parallel to this, a variance vector set, FVC, is created in Section 2.2, which is used in Section 2.5 to generate indications of the uncertainty in the identification of the architecture structure.

The outcome is thus an architecture structure, how much better suited the problem than its competitors this structure is, and how certain we are of the results.

3. EXAMPLE OF USAGE

In order to illustrate the method described in this paper, we present a fictive case in this section. We do this to give a more comprehensive presentation of how the method can be used. No importance should thus be placed on the outcome of this example, as it is how the method is applied that is the focus of the example.

The problem setting is that our fictive company is in the process of creating a new product. The product to create is a component for a telephony system as used as an example in [12]. This system is a private branch exchange (PABX) for a small company with about 50 employees. Below, we describe how this company applies our method when identifying an architecture structure for this PABX system.

3.1 Step 1: Identify Candidates

For this software component for the PABX system our company have a requirements specification, including a list of quality attribute requirements. The quality attributes are categorized into:

- Performance (containing requirements on e.g. maximum response time during normal operation and data throughput during normal operation)
- Reliability (containing requirements on e.g. uptime and maximum number of allowed missed calls during various situations)
- Maintainability (containing requirements on e.g. the maximum allowable time to conduct corrective, adaptive and perfective maintenance)
- Portability (containing requirements on e.g. the maximum number of modifications to change to a new hardware)

However, they are unsure which architectural structure is best suited for the product. They have developed a few architecture candidates using the Hofmeister et al. design method [6], but cannot decide which is the most suitable. This because they do not know how well the different architecture structures developed support different quality attributes. Nor do they know which of the quality attributes in the product to build is most important to focus on, and what effect this would have on the other quality attributes. They can choose from the following architectural structures:

- Solution A: Centralized solution based on a Microkernel [4].

Table 16. Example FQA and FAS

		Solution A	Solution B	Solution C	Solution D	FQA-Sum
Performance	FQA	0.317	0.074	0.509	0.1	1
	FAS	0.565	0.085	0.235	0.134	
Reliability	FQA	0.05	0.222	0.142	0.586	1
	FAS	0.269	0.284	0.139	0.62	
Maintainability	FQA	0.065	0.576	0.105	0.254	1
	FAS	0.075	0.571	0.081	0.183	
Portability	FQA	0.087	0.087	0.665	0.16	1
	FAS	0.091	0.059	0.545	0.063	
FAS-Sum		1	1	1	1	

Table 17. Example FQAr

	Solution A	Solution B	Solution C	Solution D	Sum
Performance	0.332	0.084	0.437	0.147	1
Reliability	0.079	0.209	0.144	0.566	1
Maintainability	0.055	0.582	0.114	0.248	1
Portability	0.073	0.076	0.724	0.127	1

- Solution B: Distributed solution based on a Microkernel [4].
- Solution C: Centralized solution based on a Blackboard [4].
- Solution D: Distributed solution based on a Blackboard [4].

The objective is to identify the best possible structure using the proposed method, which recommend a structure based on quality attributes.

3.2 Step 2: Create Framework

Using a multi-criteria decision process [1], e.g. AHP [17], the software developers create two tables corresponding to the FQA and FAS presented in Section 2.2. These two tables are presented together in Table 16.

Although the values are created using subjective assessments of the qualities of software architecture structures, it is our belief that the subjective judgements of a set of professional software developers is an accurate representation of the actual qualities that said architecture structures exhibit.

We would also like to point out that the values (albeit not the names of the architecture structures or the quality attributes) are in fact part of the results from a study [19] which we have conducted, and represents the opinions of a single participant in this study. We are currently conducting research on how separate participants' opinions can be combined into a consensus framework.

Using these two tables and the process described in Section 2.2.2, the FQAr is created, as presented in Table 17. This is the table that is used in later stages.

Furthermore, a variance vector set FVC is created according to Section 2.2.3, presented in Table 18.

3.3 Step 3: Prioritize Quality Attributes

Next, our fictive company let the stakeholders (in this case, the stakeholders belong to the same company) prioritize the quality attributes for the product to build. The resulting vector, corresponding to the PQA-vector introduced in Section 2.3, is presented in Table 19. This vector is created using the same multi-criteria decision process as was used to create the FQA and FAS. As can be seen, reliability is prioritized as being almost three times as important as the second most important quality attribute, performance, which in turn is twice as important as maintainability. Portability, finally, is the least important quality attribute.

Table 18. FVC

	S. A	S. B	S. C	S. D
Performance	0.0062	0.0003	0.0136	0.0047
Reliability	0.0018	0.0004	0.0028	0.0054
Maintainability	0.0003	0.0003	0.0016	0.0018
Portability	0.0007	0.0003	0.0078	0.0024

Table 19. PQA vector for system to design

Attribute	Priority
Performance	0.219
Reliability	0.637
Maintainability	0.106
Portability	0.038

Table 20. Value of each architectural structure

Architectural Structure	Value	Uncertainty
Solution A	0.132	0.00103
Solution B	0.217	0.00018
Solution C	0.227	0.00181
Solution D	0.424	0.00243

3.4 Step 4: Suggest Architectural Structure

Using the formula in Section 2.4, a value for each architectural structure is obtained as presented in Table 20. The architecture structure named Solution D has the highest value, and is hence identified as the architectural structure upon which to base the product. That Solution D "wins" is mainly because it has a very high value for reliability in the FQAr, i.e. a very high support for reliability compared to the other architecture structures, and this is the quality attribute that was by far the most prioritized quality attribute.

3.5 Step 5: Determine Uncertainty

The variance is calculated according to Section 2.5 (the values are presented in Table 20), and is found to be rather low, which strengthens the certainty that this is the correct choice. Moreover, Solution D gets a nomination value (i.e. not the uncertainty value but the strength by which our method nominates Solution D) that is almost twice that of its competitors, which also strengthens the decision. However, it also gets the highest uncertainty value, which lessens the determination that this is the best architecture slightly. We have not yet conducted any research on how to interpret high uncertainty values, or what constitutes a high uncertainty value.

4. DISCUSSION AND EXTENSIONS

A key element of the method in this paper is the framework, as the method depends on whether it is at all possible to create this framework. We have conducted an experiment where a framework for a particular set of software architectures and a particular set of quality attributes is used [19]. In this experiment we use five of the architectural patterns in Buschmann et al. 1996 [4] and the quality attributes in ISO 9126 [7], together with the AHP method [17].

The next step is to conduct case studies to verify the contents of such a vector set, and possibly to extend the formula presented in Section 2.4, if it proves to be too simple a model. We are currently designing such a case study, in which we intend to let a set of software developers with more than ten years of experience within a particular domain first develop a set of architecture candidates for a system in this domain and then letting them use the decision support method in this paper and match the outcome of this with the intuitive decision of the software developers.

As presented in Section 2.2, other ways to use the created framework include using it to study similarities and differences between different structures, identify strengths and weaknesses of different structures, to apply the framework in the context of assessment and evaluation of structures, and to use the framework in software evolution.

Moreover, we intend to further explore the implications of variance. For example, it is not clear what to do when two architectural structures score very similar, or equal, values when applying the formula in Section 2.4.

5. CONCLUSIONS

In this paper we present a method to further the understanding of the benefits and liabilities of different architecture structures with respect to quality attributes. Moreover, the method can be used to indicate which of the architecture structures that best suits the quality requirements of a given software system.

The method takes as input a set of quality requirements for a software system, and a set of architectural structures. During the process two sets of vectors, containing (a) a comparison of different architectural structures with respect to different quality attributes, and (b) a comparison of different quality attributes with respect to different architectural structures, are created and further refined.

The use of the method produces a list of values for the different candidate architectural structures, of which the one obtaining the highest value indicates the most suitable for the system to construct.

The method, and our use of AHP to obtain the initial values, is a way to systematically quantify the experience of the developers, as it is the subjective judgements of the developers that are asked for in step 2 (create method framework) and 3 (prioritize quality attributes) of the method. Two of the major benefits of the method is that it forces developers to systematically consider all possible combinations and that it clearly indicates where the developers are disagreeing. These disagreements will lead to discussions and, eventually, a better understanding of the problem and hopefully an agreement among the developers.

To summarize, the proposed method enables software designers to take into account relevant quality attributes for a system and evaluate these against all software architecture structure candidates for the system. The architecture structure recommended by the method is the one that, according to the developers, best meet the quality attribute requirements for the system. This can then either be used to actually create the software system accordingly, or by directing evolution work to aspire towards the nominated architecture, in order to work against software aging.

References

[1] D.R. Anderson, D.J. Sweeney, T.A. Williams, *"An Introduction to Management Science: Quantitative Approaches to Decision Making"*, South Western College Publishing, Cincinnati Ohio, 2000.

[2] L. Bass, P. Clements, R. Kazman, *"Software Architecture in Practice"*, Addison-Wesley Publishing Co., Reading MA, 1998.

[3] J. Bosch, *"Design & Use of Software Architectures - Adopting and Evolving a Product Line Approach"*, Addison-Wesley, Harlow UK, 2000.

[4] Buschmann, F., Jäkel, C., Meunier, R., Rohnert, H., Stahl, M., *"Pattern-Oriented Software Architecture - A System of Patterns"*, John Wiley & Sons, Chichester UK, 1996.

[5] L. Chung, B.A. Nixon, E. Yu, J. Mylopoluos, *"Non-Functional Requirements in Software Engineering"*, Kluwer Academic Publishers, Dordrecht, the Netherlands, 2000.

[6] C. Hofmeister, R. Nord, D. Soni, *"Applied Software Architecture"*, Addison-Wesley, Reading MA., 2000.

[7] *"Software Qualities"*, ISO/IEC FDIS 9126-1:2000(E).

[8] I. Jacobson, G. Booch, J. Rumbaugh, *"The Unified Software Development Process"*, Addison-Wesley, Reading MA, 1999.

[9] E. Johansson, M. Höst, A. Wesslén, L. Bratthall, "The Importance of Quality Requirements in Software Platform Development - A Survey", in *Proceedings of HICSS-34*, Maui Hawaii, January 2001.

[10] E. Johansson, M. Höst, "Tracking Degradation in Software Product Lines through Measurement of Design Rule Violations", to appear in *Proceedings of the 14th International Conference on Software Engineering and Knowledge Engineering (SEKE)*, Italy, July 2002.

[11] J. Karlsson and K. Ryan, "A Cost-Value Approach for Prioritizing Requirements", in *IEEE Software* **14** (5):67–74, 1997.

[12] J. Karlsson, C. Wohlin and B. Regnell, "An Evaluation of Methods for Prioritizing Software Requirements", in *Information and Software Technology*, **39**(14-15):938-947, 1998.

[13] R. Kazman, M. Barbacci, M. Klein, S. J. Carrihe, S.G. Woods, "Experiences with performing Architecture Tradeoff Analysis", in *Proceedings of ICSE'99*, Los Angeles CA., pp. 54-63, May 1999.

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

[15] G. Kotonya, I. Sommerville, *"Requirements Engineering"*, John Wiley & Sons, Chichester UK, 1998.

[16] D.L. Parnas, "Software Aging", in *Proceedings of the 16th International Conference on Software Engineering*, IEEE Computer Society Press, Los Alamitos CA, pp. 279-287, 1994.

[17] T. L. Saaty, *"The Analytic Hierarchy Process"*, McGraw Hill, Inc., New York NY, 1980.

[18] M. Shepperd, S. Barker, M. Aylett, "The Analytic Hierarchy Process and almost Dataless Prediction", in *Project Control for Software Quality - Proceedings of ESCOM-SCOPE 99*, R.J. Kusters, A. Cowderoy, F.J. Heemstra, E.P.W.M. van Weenendaal (eds), Shaker Publishing BV, Maastricht the Netherlands, 1999.

[19] M. Svahnberg, C. Wohlin, "Evaluation of Software Quality Aspects for Architectural Structures using the Analythical Hierarchy Process", submitted, 2002.