TAMPERE UNIVERSITY OF TECHNOLOGY
Department of Information Technology

ESSI LAHTINEN

SCENARIO-BASED ASSESSMENT OF SOFTWARE ARCHITECTURES

Master of Science Thesis

Subject approved by the department council
March 13, 2002
Examiners: Prof. Kai Koskimies (TUT)
Dr. Maarit Harsu (TUT)

Foreword

I have written this thesis while working at the Software Systems Laboratory at the Tampere University of Technology during the years 2001–2002. The thesis was written as a part of the research project Archimedes, which has been funded by TEKES and several industrial partners.

I would like to thank the examiners, Professor Kai Koskimies and Dr. Maarit Harsu, for their expert advice and support.

I would also like to thank my colleague Tommi Myllymäki for advising me on the field of software architectures and my other colleagues Juhana Helovuo, Terhi Kilamo, Johannes Koskinen and Antti Puhakka for their technical support. Lastly I would like to thank my friend Mari Vesanummi for her help in the linguistic part of the work and my friend Janne Manninen for all the other support I got during writing this thesis.

Tampere, April 11, 2002

Essi Lahtinen
Insinöörinkatu 24 A 13
33720 Tampere

<essi@cs.tut.fi>
<URL:http://www.cs.tut.fi/~essi/>

TAMPERE UNIVERSITY OF TECHNOLOGY
Department of Information Technology
Institute of Software Systems

LAHTINEN, ESSI: Scenario-based Assessment of Software Architectures

Master of Science Thesis, 48 pages
April 2002

Examiners: Prof. Kai Koskimies, Dr. Maarit Harsu
Funding: TEKES, Almare, Ionific, Nokia Networks, Nokia Mobile Phones, Plustech and Profit
Keywords: software architectures, architectural assessment, scenario-based assessment, product-line architectures

Abstract

Software architecture is a product of the first design phase of the software development process, so assessing an architecture can be done early in the design process. Thus, the problems perceived by architectural assessment are usually easy to correct and the assessment is relatively inexpensive.

There are many different techniques for architectural assessment. Scenario-based assessment is probably the most common because it is a general-purpose technique and easy to use. Scenario-based assessment is a questioning technique that assesses the software architecture by developing scenarios of the software development and usage and applying them to the architecture.

There are two usual methods for scenario-based assessment: the software architecture analysis method (SAAM) and the architecture trade-off analysis method (ATAM). They both have similar approaches to the assessment but the ATAM process is a little heavier than the SAAM process.

The topic of this thesis is assessing the product-line architecture of a warehouse management system by using the SAAM. The case study concentrates on assessing the modifiability of the product line but also reusability and learnability are taken into account. The problems that arise with the method and its usage in each of the assessment steps are analyzed. Ideas to improve the assessment method, the team, and the conditions under which the analysis is performed are also represented.

TAMPEREEN TEKNILLINEN KORKEAKOULU
Tietotekniikan osasto
Ohjelmistotekniikka

LAHTINEN, ESSI: Ohjelmistoarkkitehtuurien skenaariopohjainen arviointi

Diplomityö, 48 s.
Huhtikuu 2002

Tarkastajat: prof. Kai Koskimies, tri Maarit Harsu
Rahoittajat: TEKES, Almare, Ionific, Nokia Networks, Nokia Mobile Phones, Plustech ja Profit
Avainsanat: ohjelmistoarkkitehtuuri, arkkitehtuurin arviointi, skenaariopohjainen arviointi, tuoterunkoarkkitehtuurit, software architectures, architectural assessment, scenario-based assessment, product-line architectures

Tiivistelmä

Ohjelmistoarkkitehtuuri on ohjelmistoprojektin ensimmäinen suunnittelutulos, joten arkkitehtuurin arviointi voidaan tehdä aikaisessa vaiheessa projektia. Näin ollen arkkitehtuurin arvioinnissa löydetyt virheet ovat yleensä helppoja korjata. Lisäksi arvioinnin suorittaminen on suhteellisen edullista.

Arkkitehtuurin arviointiin on useita eri tekniikoita, joista skenaariopohjainen arviointi on todennäköisesti yleisin, koska se sopii useisiin eri tarkoituksiin ja on helppokäyttöinen. Skenaariopohjainen arviointi on kyselytekniikka, joka arvioi ohjelmistoarkkitehtuuria kehittämällä skenaarioita ohjelmiston kehityksestä ja käytöstä ja soveltamalla niitä kyseiseen arkkitehtuuriin.

Skenaariopohjaiseen arviointiin on olemassa kaksi yleistä arviointimenetelmää: SAAM (software architecture analysis method) ja ATAM (architecture trade-off analysis method). Molemmat suorittavat arvioinnin hyvin samantyylisesti, mutta ATAM-arvionti on hieman raskaampi kuin SAAM-arviointi.

Tässä diplomityössä tarkastellaan SAAM-menetelmällä tehtyä varastonhallintajärjestelmän tuoterunkoarkkitehtuurin arviointia. Tapausesimerkki keskittyy järjestelmän muunneltavuuteen, mutta myös uudellenkäytettävyys ja ohjelman helppokäyttöisyys otetaan huomioon. Jokainen arvioinnin osa analysoidaan erityisesti kiinnittäen huomiota menetelmästä ja sen käytöstä johtuviin ongelmiin. Lisäksi esitetään parannusehdotuksia arviointimenetelmälle, arviointiryhmälle ja olosuhteille, joissa arviointi suoritetaan.

Contents

Terms

ATAM Architecture tradeoff analysis method

ERP Enterprise resource planing

JDBC Java database connectivity

MVC Model-View-Controller pattern

SAAM Software architecture analysis method

TCP/IP Transmission control protocol / Internet protocol

WMS Warehouse management system

XML External mark-up language

1 Introduction

The *architecture* of a software system includes a description of the components of the system and the interactions of these components. The description of the components usually defines what services the components provide, how the components are roughly performed, how the faults and shared resources are handled, and so on [Bass et al., 1998]. In brief the architecture is a way to make people understand the system. It is an abstraction of the system.

Since the software systems become larger and more complex the software architectures are becoming more and more important in software development. There are three reasons why the architecture is especially important in large and complex systems. First of all, it helps the communication between the people involved with the development of the system and its users. Secondly, it contains the design decisions made early in the development process i.e. the things that will remain during the rest of the development process. Thirdly, it is a model of the system that can be reused and transferred to other contents. [Clements et al., 2001]

As a consequence of the software becoming larger and more complex, the benefits of reusing the components and architectures become larger and more desired. A set of systems that share a common software architecture and a set of reusable components is called a *software product line*. A successful product line is based on an architecture that maximizes the benefits of the commonalities between the products of the product line and allows the designed variation of the products without complications. [Bosch, 2000]

To find such an architecture for the product line, the qualities of each product have to be assessed in a systematic way. The assessment should not only concentrate on the requirements of a single product but on the generality and adaptability of the whole product-line domain. [Dobrica et al., 2000]

This thesis will concentrate on the assessment of software systems, especially from the product-line architecture's point of view. The emphasis is on scenario-based assessment which can be easily adapted to an architecture that is not too clearly described or has to be assessed already at an early stage of development. Furthermore, other techniques are considered and introduced at a more general level to get a good idea of the field of architectural assessment.

The thesis begins with an overview of architectural assessment in Chapter 2 which also introduces briefly few of the existing assessment techniques. Chapter 3 concentrates particularly on the scenario-based assessment technique to get a view of the theoretical side before applying the theory in the practice.

The product-line architectures are mainly taken into account in the case study section in Chapter 4. The case study introduces a warehouse management system architecture and applies a scenario-based assessment method to it.

Chapter 5 analyses the assessment and introduces the difficulties of the case study more clearly, to get an idea of the things that could be subject to more accurate research. The last part of the thesis is the summary and the development ideas in Chapter 6.

2 Architectural assessment

Very often a software is first designed and developed and only the complete product is then analyzed. This is a waste of resources because a lot of effort is put into developing the system without the system necessarily being good. Instead of leaving the assessment in the end, the architecture should be assessed already before the product exists. [Bosch, 2000]

If the problems are found early enough in the development process, they are easier to correct. Thus, the assessment of the architecture is more cost-effective if it is done at an early stage. The quality of the software can not be increased later, but it has to be built during the system development. Thus, the early assessment is very useful for finding out what the system will be like. [Bass et al., 1998]

Even if the architecture can be assessed already before it is fully completed, the early assessment is not the only use of architectural assessment. It can also be done for a completed system. Most likely this will be useful when an organization inherits a system, that is not well known among them, and decides to incorporate it as a part of their own system. In this case the assessment will help to understand the system and to verify that it will meet the requirements, instead of making architectural decisions as in the early assessment. [Clements et al., 2001]

The assessment is pull through by the assessment team. The assessment team should not be part of the project team, because the developers of the software lack the objective insight of the architecture, that outsiders have. In addition to the assessment team, the stakeholders take part in the assessment. The stakeholders include all the people, who have an interest in the architecture or the system built of the architecture, for example the customer, the end user, the project team (architects, coders, integrators, testers), the maintainers and the marketing personnel.

2.1 Why assess the architecture?

Architecture is the earliest stage of the software development, where quality requirements can be stated, because it is the product of the first design phase. Thus it will be the most inexpensive state for possible changes. Architecture also has a big effect on the system, so architectural assessment is the chance for the most effective improvements. The qualities that are most difficult to satisfy, such as modifiability, depend especially on the architecture. [Clements et al., 2001]

The whole development process of the system is structured according to the architecture so major changes in the architecture in the middle of the project can cause problems even at managing the project. Architectural assessment is an inexpensive way to avoid this. [Clements et al., 2001]

However, the architecture cannot guarantee that the system will be successful. A good architecture can be ruined by poor component design, implementation or testing. But in any case the other development phases cannot compensate a poor architectural design, and thus, the architectural assessment is important even if it does not tell the whole truth about the quality of the system. [Bass et al., 1998]

2.2 Quality attributes

A good architecture means very different things in different contexts so the quality attributes of the architecture have to be defined carefully before the architectural assessment. The quality attributes must always be defined within a context.

It is not enough that the qualities desired for the system are named, since most of them are too complex to be described by just one word. Instead, the value of the quality attribute has to be defined in a trivial way. For example ”easily modifiable” describes much less than ”adding a function can be done by modifying only one component” or ”changing the data base supplier can be done by changing less than 200 lines of code”. [Bass et al., 1998]

The quality attributes of the system can be divided into two categories: the attributes observable during the execution of the system (for example performance, security, availability, functionality, usability) and the ones not observable during the execution (for example modifiability, portability, reusability, integrability, testability). In addition to the qualities of the system, there are also qualities of the software development process that depend much on the architecture (for example time to market, and development costs) and qualities of the architecture (conceptual integrity, correctness, completeness) that can be measured at the architectural assessment. [Bass et al., 1998]

All the qualities of the system are not architectural. These qualities, for instance usability, have to be considered in all the phases of the software development. As an example of highly architectural quality we take modifiability. It depends mainly on the division of the

components. There are also qualities that are both architectural and non-architectural, such as performance. [Bass et al., 1998]

Finally, all the decisions about the quality attributes are compromises. It is impossible to maximize a quality without causing effects on other qualities of the system. [Bass et al., 1998]

2.3 Architectural assessment techniques

This section is written mainly according to [Bosch, 2000].

The goals of an architectural assessment can vary according to what the situation is before the assessment. Sometimes the point is to decide which of two or more architectures is more suitable and sometimes there is a need to verify that the architecture satisfies the qualities.

Qualitative assessment techniques answer whether an architecture satisfies a quality attribute or not. They can for instance compare architectures and reveal, which of them satisfies the requirements best. The disadvantage is that the techniques do not tell whether the best architecture is slightly or considerably better of a certain quality. Sometimes the qualitative assessment gives too little information, for example when the assessment concentrates on two qualities and one candidate architecture is better in the first where as the other architecture in the second. In many cases the qualitative assessment is though enough.

The *quantitative assessment* is much more difficult to carry out than the qualitative, because it should also give quantitative information about the quality attributes, for example about performance in transactions per time unit or modifiability in changed lines of code. Usually this is difficult because the assessment is done already before the system exists, but if the quantitative assessment is possible, it can be very useful. The weak point of this kind of assessment is that the values may be difficult to compare without theoretical maximum or minimum values.

When performing a high quality assessment the theoretical maximums and minimums would be defined and the gap between them would be used for deciding whether the architecture should still be developed. The techniques currently available can not do much about this. Neither are the quantitative techniques yet as accurate as they could be. Because of the assessment costs at present most of the assessments are performed in the qualitative techniques.

Architectural assessment methods can also be divided into two categories according to how they work: questioning and measuring techniques. *Questioning techniques* can assess any given quality of an architecture. They assess architectures by generating questions about their qualities. Examples of questioning techniques are scenarios, questionnaires and checklists. *Measuring techniques* are more restricted than questioning techniques, because they only assess specific software qualities. They make quantitative measures of these qualities. Measuring techniques include for example metrics, simulations, prototypes and experiments.

The best assessment results are usually gained when both questioning and measuring techniques are used. Usually measuring techniques are not so good for getting a concept of the whole system, but they might be good for finding out more about the issues raised in the questioning assessment.

Some existing assessment techniques are presented in the following subsections according to [Bass et al., 1998].

2.3.1 Scenario-based assessment

The idea of scenario-based assessment is to describe different kinds of situations where the stakeholders interact with the system. These situation descriptions are called scenarios. The scenarios are then applied to the architecture and this way assessed as realistically as possible. Scenarios are, as it were, questions about the system and scenario-based assessment is a questioning technique. It can be used for qualitative assessment.

Scenario-based assessment is suitable for assessing the architecture already at a very early stage of the architecture development process since the scenarios can be chosen so that only the main structure of the architecture is needed for the assessment. Scenario-based assessment is also relatively inexpensive and quick and that is why it is probably the most common way of assessing architectures.

2.3.2 Questionnaire-based assessment

A questionnaire signifies a list of common questions that can be applied to almost any architecture. The questions concern for example the way the architecture was developed and documented and the features of the architecture. The aim of the questionnaire is to produce a detailed description and definition of the domain so that all its architectures can be assessed with the same questionnaire.

Before the assessment the assessment team will decide which answers are of preference and will look for an architecture that has most of the desired features. If the questions are too common for a particular architecture, they will focus the question as long as it is detailed enough to give an answer.

2.3.3 Checklist-based assessment

A checklist is a set of questions that are more detailed than the questions in the questionnaire-based assessment. It is usually based on a long experience in assessing applications from a certain domain, because it is not easy to develop a competent set of such questions. Checklists are usually focused on a certain quality attribute of the system so one checklist is not enough for a complete assessment.

Questionnaire-based and checklist-based assessments seem to be very similar to scenario-based assessment. The biggest difference is that scenarios are developed for a specific system and questionnaires and checklists are general. An example of a scenario could be ”What happens when a database error occurs?” and the corresponding checklist question could be ”Is there a retry mechanism and reporting to the user when a function fails?”. The scenario points out a certain error of the application and the assessment team has to know what the desired behavior is. The checklist just wants to point out whether the feature exists or not.

When assessing many different systems from the same domain, the scenarios that keep on occurring in most of the assessment can form a questionnaire or a checklist. This way the questionnaires and checklists can be called more mature than scenarios.

2.3.4 Simulation-based assessment

Simulation-based assessment is explained according to [Bosch, 2000].

As its name reveals the idea of simulation-based assessment is to simulate the system. The point is to give a more dynamic alternative to assess the system compared to the scenario-based assessment that is rather static since there is no real system whose behavior could be observed.

The assessment of course requires a simulation model both of the environment and the architectural components. To benefit the assessment the simulation model naturally needs to be much simpler than the real system, so the system has to be simplified radically. It can be difficult to decide which attributes of the real world should be discarded as meaningless. The simulation model of the components should be rather easy to develop according to the architecture presentation.

After the model of the system is developed, a profile for each quality attribute is needed. The scenarios of the profile are simulated and the data collected. The final assessment is fairly similar to that of the scenario-based assessment.

Simulation-based assessment requires a little more work than for example scenarios, though it can also be more beneficial because it forces the architects to think about the interfaces of the components very carefully. It is impossible to build a simulation model without them. The executable model of the system can be very useful as well at other stages of the software development.

Simulations are useful primarily for assessing qualities observable at the run time but also qualities concerning the development of the system can be assessed. It has proved to be useful for assessing embedded systems in particular, especially with requirements on availability and safety.

One alternative to the simulation-based assessment is mathematical modeling. It is very similar to simulation-based assessment. The only difference is that the model is not a real version of the software but a theoretical model expressed with mathematical means.

2.3.5 Metrics-based assessment

Like simulations, metrics are measuring techniques so they also need a model of the architecture. For metrics the model does not always need to be executable like the simulation model. Depending on the metrics, a written model of the structure of the components and the call graphs may be sufficient.

Metrics are quantitative measurements done on particular architectural points such as fan-in (the number of other modules that call the module) or fan-out (the number of other modules being called by the module). Metrics-based assessment is good for defining the overall complexity of the system and it can be used for finding the interfaces that are likely to be changed or where the changes will be difficult if they occur.

2.4 The benefits and problems of architectural assessment

This section is presented according to [Clements et al., 2001].

It is obvious that by assessing the architecture the architectural problems will be discovered and they are cheaper to correct, or if none are found the confidence in the architecture increases. However, the biggest benefit gained from the assessment is probably the fact that all members of the development team will understand the architecture and even those who already did, will have to rethink it through.

There are some other benefits too, even if they do not arise in every assessment. The assessment draws the stakeholders together for a while and the established contacts may be helpful at the other stages of the development process. Furthermore when all relevant people are listening, the requirements will be stated more carefully and the assessment team might capture things they did not find in the requirement documents.

There will always occur conflicts within the requirements, since the architecture will usually not be able to satisfy all of the requirements. When all the people involved in the process prioritize the requirements jointly, the architect will receive clear instructions on what are the most important ones. This will help the architect a great deal in making the decisions and all the participants will understand the ground of the decisions better.

The documentation of the architecture will be better because it is needed already at the beginning of the project. Since the architects have already had to explain the architecture in the assessment they are prepared to explain it to people not familiar with it also later during the process.

The assessment can also produce new ideas on how to benefit from the project because there is a larger team from different development areas present in the assessment. There might for example be people with ideas on how the system or its components could be reused, that the architects did not come up with.

In organizations where assessments are done regularly, the architecture practices have been improved. The organizations learn to formulate the beneficial questions and bring up the problematic issues. People learn to prepare these issues already in designing the architecture and write the documentation accordingly. The performance of the assessment is maximized in the long run and this leads to high quality architectural design.

Even if the assessment is relatively inexpensive compared to the costs of radical architecture changes, it might be too expensive for small organizations. A full assessment usually delays the project approximately 3 days, but since a number of people have to be present at certain phases of the assessment it can become expensive. Fortunately not all the stakeholders are usually employed by the organization developing the system. Since they are the ones who will benefit from the system being as required they are usually willing to spare their time.

3 Scenario-based assessment

This chapter has been written mainly according to [Bosch, 2000].

Scenario-based assessment does not simply assess architectures on one scale, the good architectures and the bad architectures, but it takes different points of view to the architecture and analyses on each of these scales. Thus instead of producing a single architectural metric, scenario-based assessment produces a collection of small metrics.

The metrics are developed according to the scenarios that represent interactions of all the stakeholders and the system. The scenarios are real life situations that may occur during the lifetime of the product. Thus the scenario-based assessment techniques do not leave the assessed quality attributes out of context, but they give a real situation to assess even if the product is not finished yet.

It is possible to create different scenarios for different stakeholder requirements. Thus, scenario-based assessment is good for paying regard to all the stakeholders and taking different points of view to the system.

Scenario-based assessment is a very general-purpose assessment technique since it is possible to be used for many different kinds of quality attributes. It has been found particularly useful for assessing the architectural qualities essential in the developing the architecture such as maintainability.

The effectiveness of scenario-based assessment depends on the representativeness of the scenarios. It is impossible to make accurate assessments without accurate scenarios. Thus choosing the scenarios is one of the most important parts of scenario-based assessment.

3.1 Scenarios and profiles

Before assessing the architecture with the scenarios, one needs to define a profile of the scenarios for each quality attribute to be assessed. A profile is a set of scenarios relevant to the quality attribute.

A scenario is a bit similar to a use case but it is not limited to assessing the functionality, i.e. the runtime behavior of the system. Instead it can describe interactions with all the stakeholders, for example making modifications to the system or estimating the time-to-market of the system. One scenario describes briefly one interaction of one stakeholder in the system, although it represents an entire class of scenarios. The scenarios that cause exactly the same kind of response from the system are of same class.

Sets of scenarios can be divided into two categories: complete and selected sets. A complete set includes all the scenarios relevant to the quality attribute it is related to i.e. one scenario of each class of scenarios related to the quality attribute. Based on a complete set of scenarios a software architect is able to perform an assessment about the quality attribute. A complete set is thus called the profile of the quality attribute.

If the system is large or the quality attribute ambiguous, it is impossible to define complete sets of scenarios. For example a set of all the change scenarios of a system is infinite since there is an infinite number of different kinds of changes that could be performed in a system.

The scenario sets that are not complete are selected. The scenarios for a well defined selected profile are chosen randomly and thus represent a whole selection of existing scenarios. It is problematic to make a truly random profile since it is very likely there will be some structure, i.e. certain kinds of scenarios, in the scenario selection, if they are chosen by a human being. This is why the selection of the scenarios has to be done particularly carefully.

To be able to use a non-random set of scenarios as a profile, one needs to eliminate the structure of the set as much as possible. To achieve this, one needs to do the selection process in two steps. First define the scenario categories that represent the whole population of the scenarios. This means dividing the population into groups of similar kinds of scenarios. Then do the selection of scenarios within the categories. This does not mean that the selection would be random, but the randomness is still increased and to increase it even more so, one can divide the categories into subcategories. It is a weakness not to be able to provide a random set of scenarios, but the option not to use scenario-based assessment at all is worse, since scenario-based assessment is very useful at the beginning of the architecture design process and that is exactly when assessment is needed the most.

3.2 Scenario-based assessment process

Defining a certain structure for the assessment process makes the assessment repeatable and it makes sure that all the important questions are stated and all the weak points found. Thus a well organized assessment follows the structure of a named assessment process.

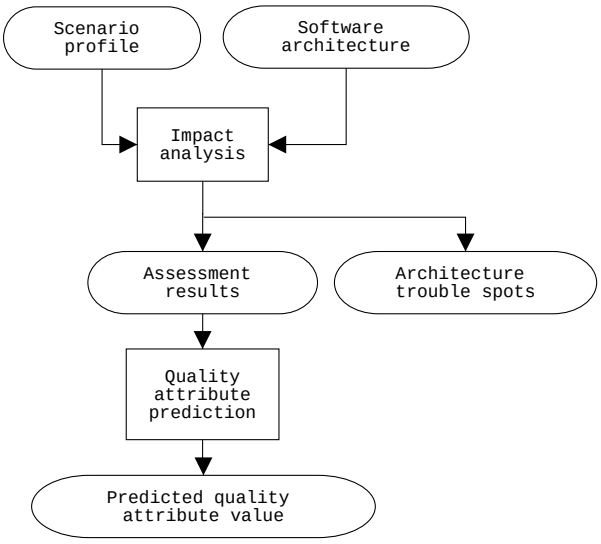

Figure 3.1: Scenario-based assessment process.

The scenario-based assessment methods can be used both for comparing architectures with each other (relative assessment, comparative assessment) and for an absolute assessment of one architecture. The assessment process varies a little depending on whether it concerns relative or absolute assessment. We will first consider the absolute assessment process.

The absolute assessment process of an architecture is presented in Figure 3.1. There are two main steps: impact analysis and quality attribute prediction. The assessment steps are presented in the rectangles and the inputs and outputs of the assessment steps in the rounded rectangles.

Impact analysis takes the scenario profile and the architecture and simply applies the architecture to the scenarios. The assessment results are written down for each scenario. The result is the concrete data that has been decided to be measured with the scenarios. For example for a change scenario the numbers of new or changed lines of code and modules would be counted or for a performance scenario the execution times estimated. In the end the measured results are summarized. If the scenario is functional or a change scenario, the impact analysis can also find trouble spots in the architecture because some actions are not applicable on all architectures or if they are, it might be too complicated to accomplish them.

The quality attribute prediction uses the assessment results the impact analysis produces and predicts the value of the quality attribute with statistics or historical data. For the change scenario the resources needed for the changes can be counted or for the performance scenario the total execution time of the system can be calculated. This outcome quality attribute value is then compared with the definition of the quality attribute.

When used for comparative assessment of two or more architectures, the scenario-based assessment will not need as much quantitative information as when used for absolute assessment, but it will also not produce that accurate estimates on the quality attribute value. This is because the impact analysis results are collected on a scale of for example ++, +, 0, -

and - -. When all the scenarios have been performed for all the architectures being assessed, the architect can choose the most suitable architecture by the scores given for each architecture. Making the choice might still be difficult according to these scores because of the lack of the quantitative information about the quality attributes for each scenario.

3.3 Scenario-based assessment methods

As examples of scenario-based assessment methods software architecture analysis method (SAAM) and architecture tradeoff analysis method (ATAM) are introduced in the following subsections.

3.3.1 Software architecture analysis method (SAAM)

SAAM analysis can be used, not only to form a scenario-based table of the characteristics of the architectures, but also to understand the architectures better, which can be very helpful in deciding on the architecture to be chosen for the system.

In this subsection we introduce the steps of SAAM assessment according to [Bass et al., 1998], but when used, the assessment need not always go through all the steps. Sometimes it is more beneficial to go through just a subset of the steps. SAAM is a flexible method so almost any subset of steps can be used alone.

Develop scenarios

The architect should make sure that the scenarios represent all different stakeholders, all different uses of the system and all different quality attributes that are to be assessed. Therefore developing the scenarios should be done very carefully by an experienced person.

Every scenario is assigned a weight according to how important the scenario is. This is especially important if the assessment is comparative. Then the weights are used in helping to decide about the ranking between the candidate architectures. To make the assessment fair, all the stakeholders should participate in weighting the scenarios.

Describe candidate architectures

If a system has to be assessed by scenario-based techniques, the architecture of the system has to be described by a notation that all the parties involved in the assessment understand. If there is not enough of documentation about the system, the architecture has to be reverse-engineered. At least the systems components (both data components and functional components) and the connections (both data flow and passing the control) between the components need to be described. It is of course better to have more description of the system.

Classify scenarios

The architecture is then applied to the scenarios i.e. the assessment team explains how the system would perform the actions of the scenario or how the system would need to be changed to be able to perform the actions.

The scenarios are classified to direct and indirect scenarios according to the behavior of the architecture. If the architecture could perform the actions required in the scenario, the scenario is direct in respect to that architecture. If the architecture needs to be changed to perform the scenario, architecture is indirect with respect to that scenario.

This classification is not so reliable because a scenario could be realizable in a complicated or poor way that actually does not suit the architecture at all or the modifications needed for fulfilling the scenario can vary a great deal. Sometimes a scenario can be almost direct, because the changes needed are minor.

Perform scenario assessments

All the indirect scenarios need to be analyzed deeper. All the changes in the architecture are listed and described and the cost and difficulty of the changes are estimated as accurately as possible.

In the end there should be a list of all the architectures, the scenarios that were direct and indirect in respect to that architecture and the corrections needed for the indirect scenarios. If the assessment is comparative, it is usually helpful to make a table of these results.

Reveal scenario interaction

If there are two indirect scenarios that require changes to one single component, the scenarios are said to interact in that component. The scenario interaction is important, because it reveals a variety of things about the responsibilities of the components. If there are scenarios that are not functionally related but are interacting, the component where the interaction takes place is computing unrelated functions. If certain components interact a lot, the interfaces of the components might not be so carefully designed and the responsibilities of the components should be revised.

By the scenario interactions one can tell about the coupling between the components. Interaction of similar scenarios denotes high cohesion and interaction of different kinds of scenarios denotes high structural complexity. Therefore the amount of scenario interaction correlates with the number of faults in the final product.

Overall assessment

In the end the assessment team should decide on the best way of dividing the system into components. This can be done by studying the results of the earlier steps considering the weights of the scenarios.

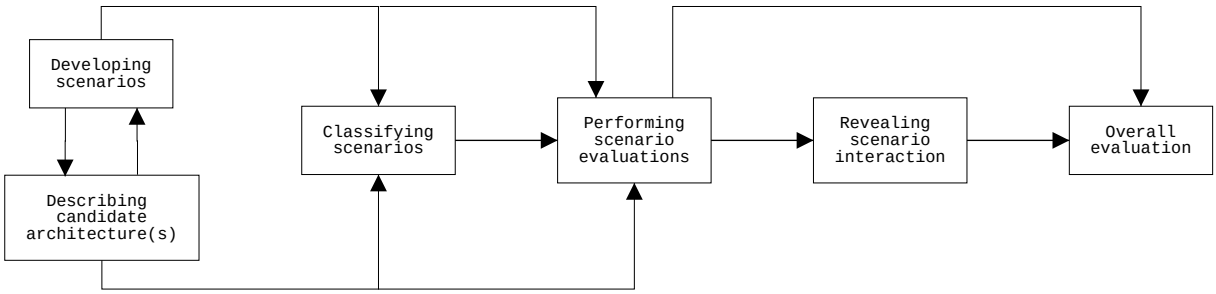

Figure 3.2: The dependencies between the steps of SAAM assessment.

The assessment team should bare in mind the whole time that the assessment steps depend on each other. If one step seems to be difficult or impossible, the earlier steps can be rethought and the problem can probably be found in the defect of one of the earlier stages. Figure 3.2 shows the dependencies between the steps of SAAM assessment.

Although in SAAM the emphasis is on assessing modifiability and functionality of the system, as a scenario-based assessment method it should be potential for assessing all other kinds of quality attributes too.

3.3.2 Architecture tradeoff analysis method (ATAM)

ATAM is a successor of SAAM. It does not only assess how the quality attributes are considered in the system, but also how they trade off with each other. This makes ATAM a heavier process than SAAM is, so the assessment team should be able to invest more time if they want to do the assessment using ATAM.

The ATAM process can be divided into four phases that arrange the steps of ATAM into a beneficial order. The steps of ATAM are presented first according to [Kazman et al., 2000].

1. Present the ATAM

All the participants of the assessment team need to understand the process, so the ATAM team has to explain how it will be done and answer questions concerning it. The presentation will include all the steps of ATAM, the techniques used in them and the outputs assumed from each step.

2. Present business drivers

The business drivers are the qualities required from the product, including the functional and architectural requirements, the technical, managerial, economical and political constraints, business goals and the stakeholders. All these need to be explained to all the participants of the assessment team.

3. Present architecture

The architecture is presented by the lead architect. It should be presented from the business drivers' point of view and all the participants of the assessment should understand it. All the details relevant to the assessment should be presented but excess information is not essential. This step should contain technical constraints: the other systems interacting with the assessed system and the architectural approaches used.

4. Identify architectural approaches

The architect identifies the architectural approaches. The aim of this step is to get the other participants of the assessment team to understand the approaches. It does not include analyzing the approaches.

5. Generate a quality attribute utility tree

The desired quality attributes of the system are defined and concretised by making scenarios of them. The scenarios also include the inputs given to the system and outputs the system should produce to the corresponding input. A priority is given to each scenario. All this is summarized by building a utility tree of the quality attributes. The utility tree is a figure of tree shape with the word ”utility” on the top, the quality attribute names under it, the subcategories of the quality attributes under them and the scenarios under them.

6. Analyze architectural approaches

The architectures that suit with the high priority scenarios are chosen to be analyzed more carefully. This analysis contains identifying the architectural risks, sensitivity points and trade-off points.

7. Brainstorm and prioritize scenarios

Not to limit the assessment to the highest priority scenarios, all the stakeholders vote for the priorities of the rest of the scenarios. According to this vote the scenarios are prioritized and a set of scenarios is chosen for the rest of the assessment.

8. Analyze architectural approaches

The architectures chosen two steps earlier are analyzed again with the larger set of scenarios. This step should confirm the earlier analysis. If new information about the architecture is found, the quality attribute utility tree generation has failed and the assessment should be done again from that step.

9. Present results

The ATAM team will perform the results based on the information collected in the earlier steps.

The steps of ATAM are usually carried out in four phases. The phases are presented according to [Clements et al., 2001].

Partnership and preparations

This phase does not start with the assessment steps but it is needed before the assessment can start. The assessment team needs to establish a partnership with the sponsor of the assessment. All the participants of the assessment need to communicate with each other and form a group. The later steps need to be prepared.

Architecture-centric assessment

This phase is supposed to collect information for performing the next phases. It consists of the assessment steps from 1 to 6 and can be run through by a smaller team. All the stakeholders need not be present.

After this phase the assessment team should know if the assessment will be beneficial and if it does not seem to be so, the larger team with all the stakeholders need not be gathered.

Stakeholder-centric assessment

This part has to be done with everyone involved with the system. It begins by an encore of the steps from 1 to 6 to the new parties of the assessment. Actually the steps from 2 to 6 need not be performed but the results of the steps have to be presented so that everyone understands them. The assessment carries on in this phase from steps 7 to 9.

Follow-up

The final report is produced and the possible follow up actions planned. The assessing organization will update its experiences.

The idea of ATAM is to reveal the risks, sensitivity points and tradeoff points by pointing out how the different architectures cope in different situations by using the scenarios and the quality attribute utility trees.

ATAM is a very general-purpose assessment method, even if the emphasis has been placed on modifiability, security, reliability and performance when developing it. Compared to

ATAM, SAAM is a little bit more detailed, so the description of the architecture needs to be more elaborate for applying SAAM. For ATAM it is sufficient that the approaches of the architecture have been chosen. For SAAM the functions of the architectural components should already have been decided. [Clements et al., 2001]

4 Case study: warehouse management system

This case study is about assessing an architecture of a warehouse management system. The idea of a warehouse management system (WMS) is to automate the routines of managing the product flow through the warehouse. A WMS is typically used by a supply chain that has to manage the product flow from manufacturers to end user, the orders from customers to suppliers and from suppliers to manufacturers and the returned products and cash flow from customers to the suppliers and manufacturers.

As an example of a WMS product line we consider Done Logistics' DoneWMS described more clearly in [Ylikangas, 2001]. Nowadays the WMSs developed by Done Logistics are only 20% to 60% reused from earlier systems but the aim is to reduce the amount of tailored components by developing the DoneWMS product-line architecture.

DoneWMS helps in minimizing the paper work and making the routine decisions in functions such as

- communication about the purchase and sales orders (connection to an enterprise resource planing (ERP) system)
- reception of the products to the warehouse and shipping them out to the customer
- shelving, storing, transferring, assembling, handling, picking and labeling the products in the warehouse (the products are mainly handled as palettes)
- inventory of the warehouse

- recording and reporting the operations and the resources.

Other features of the system are that it takes care of authentication of the users. The users of the system are classified into different categories and have different views of the system and different rights accordingly. The users mainly work in the warehouse but sometimes they have to be able to access the systems data also through the Internet so a remote connection has to be supported. The system also has to be internationalisable.

4.1 Quality attributes

The quality requirements that Done Logistics has named for their product-line architecture, are maintainability, flexibility, performance and safety [Jakovlev et al., 2001]. The driving forces of the product-line approach are reusability and modifiability [Dobrica et al., 2000] so they should be assessed as well.

There is only little difference between modifiability, maintainability and flexibility. Modifiability means that it is easy to make changes to the system quickly and inexpensively in common. Maintainability means that it is easy to make changes to the components of the system to adapt to changes of the environment. Flexibility is also very close to these two. It means that the system can easily be modified to environments that it was not originally designed to [Dobrica et al., 2000]. According to these definitions maintainability and flexibility can be considered as subcategories of modifiability.

Reusability means that the systems structure or some of its components can easily be reused in other applications in the future [Dobrica et al., 2000]. Product-line architecture is a way to support reusability. The structure of the system is the same in all the products of the product line, so assessing the reusability of the structure is not necessary. Instead we can focus on assessing the reusability of the components.

In this case study safety does not mean preventing intruders from accessing the system but preventing unqualified users from misusing the system, so we shall call it learnability for the rest of the assessment.

Performance means the system's ability to use its computational resources so that it will meet its timing requirements. In this case assessing the performance is rather difficult since there is not a very exact description of the architecture available and we cannot say much about the timing of the functions.

We choose to assess the following quality attributes:

1. Modifiability (includes maintainability and flexibility)
2. Reusability
3. Learnability (in meaning of safety)

To make the assessment sensible, the desired values of the quality attributes need to be decided next. The documentation of DoneWMS does not contain information about this, so we need to sketch it. The modifiability is good if it is easy to make modifications. So we need to describe what is the limit for easy modifications. The system is easily modifiable if the most probable changes can be done by changing only two components of the system. This should be true to at least 50% of the changes considered in the assessment. There will be a vote for the most probable changes when the scenarios are developed.

As stated earlier the reuse percentage of Done Solutions software without the product-line architecture varies between 20% and 60%. To make the product-line architecture useful this has to increase. We set the goal to 80% of reused lines of code in a typical product of the product line. By a typical product we mean a warehouse management system that represents the domain of DoneWMS. We do not consider all kinds of storage applications, because the idea of a product line is to reuse the components in the environment they were designed to.

Learnability is the most abstract of the quality attributes assessed, and setting its desired value is not very obvious. It is especially difficult because the assessment has to be done considering only the architecture. More natural would be to set a goal, for example, for the time an average person needs for understanding the functions of the system by learning to use the user interface with support. We will try to do the assessment by considering the data and the functions available on the displays of the user interfaces.

We will use SAAM in the assessment, because the description of the architecture of DoneWMS includes the descriptions of the component functionality, i.e. the material needed for SAAM. ATAM could be used as well, even if it does not require the component functionality. However, the ATAM process is more complicated than SAAM process and the assessment team is not very experienced, so we had better experiment with SAAM first.

SAAM could be applied by using only a subset of the assessment steps but since the assessment team does not have experience on how to assess with SAAM, all the steps will be performed to find out which of them are useful.

4.2 Developing the scenarios

Scenarios will be developed separately for each of the quality attributes to be assessed.

4.2.1 Modifiability

Modifiability concerns changing the system so change scenarios are most suitable for assessing it. It is impossible to develop a complete set of change scenarios for any system, so we will be satisfied with a selected set.

We will try to develop scenarios as random as possible by dividing the system into change categories first. In Table 4.1 the highest level of change categories is on the left, the subcategories of which are next to them on the right, the subcategories of the subcategories next on the right and so on. The categories are divided into smaller ones until the subcategory presents a single class of scenarios.

Table 4.1: The change categories of the WMS.

description of the category				ID
the connections to the system	changing the existing connections	changing the user interfaces	changing the displays	A
			changing the connection type	B
		changing the database	changing the content	C
			changing the supplier	D
		changing the ERP system interface	adding new communications	E
			changing the existing interface	F
		changing the operating system		G
	adding new connections to the system			H
the system itself	changing existing features	changing an existing function		I
		changing an attribute to the functions		J
	adding features	adding a new function		K
		adding an attribute to the functions		L

In the categories I, J, K and L the word ”function” means an operation the user can perform in the system, such as registration of the products or the inventory of the warehouse. The word ”attribute” means a feature of the system that affects all the functions of the system, such as the category of the user logged on. The category E is a subcategory of ”changing the existing connection” but it contains the word ”adding”. There is a disharmony, but ”adding new communications” is related to ”changing the ERP system interface” so it is actually a change.

The scenarios derived from the requirements of the system [Ylikangas, 2001] and the change categories, are in Table 4.2. There were no scenarios developed for category B (a change in the communication technology between the clients and the server) because a wide range of technologies that the DoneWMS supports is already listed by [Jakovlev et al., 2001]. Thus, a change in category B was defined as very unlikely. There are three scenarios from category C, because they were so different that it was difficult to choose which one of them represents the whole category the best. The scenario number M-13 was added, because the scenario was interesting, even though it did not seem to fit in any of the categories.

The desired value decided at the beginning of the assessment was that 50% of the scenarios, the most probable ones, only cause changes in two of the components. We should now decide which of the scenarios will be the most probable. This can be done by voting. The scenarios rated as the most probable ones are: M-1, M-2, M-5, M-8, M-10 and M-11.

Table 4.2: The change scenarios for maintainability.

category	scenario description	scenario number
A	Some new text fields have to be added to the dialog window for inserting customer information.	M-1
C	The data type for the customer ID numbers or the product ID numbers has to be expanded.	M-2
C	A new type of container has to be added parallel to palettes and parcels.	M-3
C	New attributes that do not depend on each other have to be added to an existing product (for example color and size).	M-4
D	The supplier of the database has to be changed.	M-5
E	Confirmation has to be added to the process of shipping the sales orders out form the warehouse.	M-6
F	A change has to be done in the product information.	already in M-3
G	The operating system has to be changed.	M-7
H	The system has to be adapted to an automatic warehouse.	M-8
I	The sales order function has to be changed so that the order can be taken even if there is no possibility to start picking immediately because there are not enough products for the order.	M-9
J	A new user type has to be added to the system (in addition to the main user, the operator and the shift manager).	M-10
K	Hour reporting has to be added to the system.	M-11
L	Some of the functions of the system have to be allowed only from user interfaces in the warehouse i.e. remote connections are not allowed to execute these functions.	M-12
	The possibility for changing to a different currency with a change period during which both new and old currency are accepted (a currency change in the country where the system already exists, not a new system built for different currency).	M-13

4.2.2 Reusability

In case of a product line, reusability is assessed best by developing scenarios of different products that cover the domain of the product line as completely as possible. To make sure that the whole domain is covered the variation between the products has to be as big as possible within the domain.

There are infinite number of different kinds of products that could be developed of DoneWMS so a complete set of scenarios is practically impossible to create. We will divide the scenarios into categories by considering the different variability points of DoneWMS. At least two scenarios have to represent each variability point to ensure that the whole variation will be covered in the assessment. The first variation point we consider is the size of the system measured as the number of client connected to the system. The variation will occur between a small system of only a couple of shipping clients and a large system of many different kinds of clients. The second variation point is the functional complexity of the system measured as the amount of functions available in the system. The third variation point is the operating system, the other variant operates in Windows NT and the other in Linux.

Assessing these three variability points does not mean that we need to develop two scenarios for each of them. We can develop two scenarios so that both of them represent the other bound of every variation point. The scenario descriptions are presented in Table 4.3.

Table 4.3: The use scenarios for assessing the reusability of the WMS.

Scenario description	scenario number
A small manufacturing warehouse that has only two clients, both of which at the shipping desk (no reception, because the products come straight from the production lines). No remote connections are allowed and the connection to the ERP system is minimal, because there is only one person taking care of the sales orders. The products are handled as palettes all the time (they come on palettes from the production line and are sold only as palettes) so no picking is needed. The system operates on Windows NT.	R-1
A large beverage type of warehouse with a lot of clients operating (reception, shipping and remote) and a lot of connections with the ERP system. All the functions available are implemented. Operates on Linux.	R-2

4.2.3 Learnability

Since learnability is about using the system, use scenarios are good for assessing it. However the material available does not tell much about the user interfaces of the system, so the

assessment of learnability cannot be done as systematically as the assessment of modifiability. If the display sketches were available they could be used as scenario categories. Each function could be used as a category as well, but that is somewhat too detailed in this case because we do not have the displays available for detailed assessment.

The use scenarios derived according to the requirements of the system [Ylikangas, 2001] are presented in Table 4.4. They are concentrated on situations that could cause misusing the system by an unqualified user, as stated safety in [Jakovlev et al., 2001].

Table 4.4: The use scenarios for the WMS.

Scenario description	scenario number
A receptionist has entered a wrong amount of products and needs to correct it after having confirmed the registration.	L-1
An operator has to cancel an order made earlier but doesn't know the order number.	L-2
A regular user needs to substitute the shift manager for a while and needs higher rights for the system operations for a limited time period.	L-3
An event done earlier has to be recovered from the log file.	L-4

4.3 Description of the architecture

The architecture is described according to [Jakovlev et al., 2001]. The description is not very accurate because the infrastructure of DoneWMS is classified but it should be accurate enough for a cursory assessment.

The basis of the system is a normal client server architecture. The server contains all the business logic of the system. It includes a client manager, a message queue and a handler for each function of the system. The client manager receives messages from different clients and puts them to the message queue. The message handlers receive the messages from the queue and operate with the database through the JDBC.

The displayable part of the clients are Java Server Pages, and they are connected to the server through a TCP/IP-connection. The clients are developed according to the Model-View-Controller pattern (MVC). The idea of MVC is to decouple the userinterface from the business logic of the system. The user interface presents the view part of the pattern, the business logic the model and the controller keeps updating the view when changes occur in the model. The top level of the architecture is described in Figure 4.1.

The implementation of the system will use a layer model. The main layers are the infrastructure layer, the domain-application layer and the presentation layer. The infrastructure layer will contain the persistent mechanism, the client server communication and the database connection. The domain-application layer includes all the business logic, and the presentation layer takes care of the user interfaces.

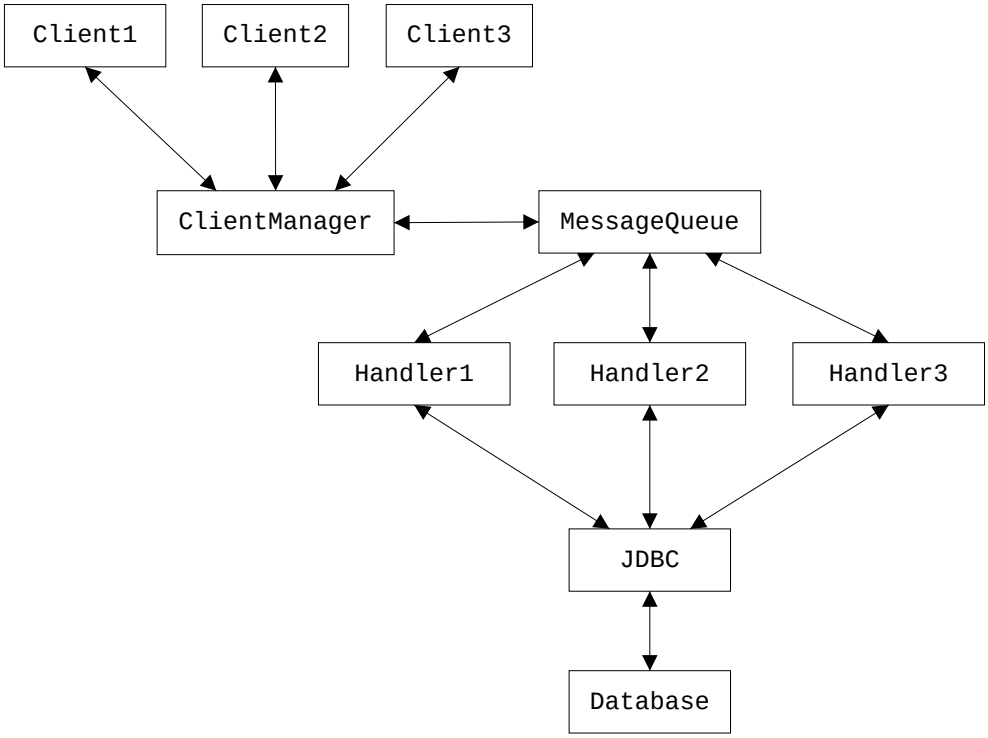

Figure 4.1: The highest level of the architecture of DoneWMS.

4.4 Developing the scenarios, second iteration

After having thought through the architectural structure of the system one has a different view of the system and rethinking the scenarios can result in new ideas. There are some more change scenarios for assessing modifiability in Table 4.5.

Table 4.5: More change scenarios for the WMS.

scenario description	scenario number
The reception of products is changed so that if the product is not yet registered in the system it will be automatically registered during the reception.	M-14
The number of clients exceed the capacity of the client manager.	M-15

Now with the increased number of scenarios we can also add one more scenario to the list of 50% of the most probable scenarios. We will add the scenario number M-15 and there will altogether be 7 scenarios on the list.

4.5 Scenario classification

The next step of the assessment is to classify the scenarios to direct and indirect ones.

4.5.1 Scenarios for modifiability

Expanding the data type for the customer ID numbers or the product ID numbers, like in scenario M-2, requires only a change of one line of code because the IDs are defined as type definitions instead of a certain number type. The change is easy to do because no changes in the interfaces that handle the ID are needed. Of course a minor change in the database tables has to be done too. The change in the scenario is not radical, therefore it can be considered direct.

Scenario M-5 is a direct scenario because the JDBC is decoupling the database from the handlers so the change can be done without modifying the system.

The rest of the scenarios seem to need larger changes so they are indirect and will be assessed more carefully.

4.5.2 Scenarios for reusability

The scenarios depict the new products to be developed of the product line. They could be considered direct, because the product line was intended for developing such systems, and thus, does not require changes. However, in order to gain results from the assessment, we will assess them anyway, because the point was to assess the reusability in the domain of the product line on a given scale.

4.5.3 Scenarios for learnability

Reading through the documentation of DoneWMS does not facilitate the definition of the learnability scenarios being direct or indirect. There is no information available about the displays at all.

This quality attribute will be left out of the assessment at this point. It could be better carried out by a person who knows the system better, or later on during the development of the product with a usability specialist. After all, learnability depends more on well designed displays with good usability than the architecture of the system.

4.6 Performing the scenario assessments

The scenarios that were not direct will now be assessed more carefully.

4.6.1 Assessing the modifiability scenarios

Each indirect scenario has to be applied to the architecture. The important thing is to describe which part of the system needs to be changed to make the scenario work so that the further assessment can be done.

If there were two different architectures to compare, it would be very important to state how much work the changes would require. This is almost impossible with such little information available about the system, so this assessment concentrates on finding the components that need to be changed.

The scenario assessments will be presented one by one:

M-1: Adding some new text fields to the dialogue window for inserting customer information should not be a big change in the format of the XML-message transporting the data, but also a change in the handler of the message needs to be done.

M-3: Adding a new type of container parallel to palettes and parcels will cause many changes. First of all, the new container type has to be added to the database, then all the displays and all the message handlers that deal with the products need modifications. The interface to the ERP has to be modified and in the worst case the ERP itself needs modifications, but the changes to the ERP are irrelevant in this assessment.

M-4: When adding attributes that do not depend on each other, such as color and size, a whole new database table needs to be added to the system, because both the attributes have to be stored for each item stored in the warehouse. Changes on the displays and the handlers related to the product data also need to be done. Changes in the ERP interface may occur as well.

M-6: Adding a confirmation to the process of shipping sales orders out from the warehouse causes a change in the handler of the shipping messages. The handler needs to call the ERP system so also the ERP interface will be changed. The display might also need slight modifications.

M-7: The layer implementation of the server enables changing the operating system quite well. The application layer is the layer nearest to the operating system and most of the changes will be done there but the infrastructure layer might need changes too. With the documentation available it is difficult to say which of the components need to be changed but at least all the components needs to be tested again in the end to verify that all the changes required are done.

M-8: Adapting the WMS to an automatic warehouse will cause a need of a whole new interface between the device drivers and the system. The drivers will mainly interact with the handlers that manage the automated operations but also the drivers will have to be able to send messages to the message queue. For example, when internal transfers of the products are done and their locations need to be updated. This means that the highest level of the architecture description will be changed.

M-9: If the sales order function has to be changed so that the order can be taken even if there is no possibility to start the picking immediately, the main changes will of course concern the handler of this function. However, an additional database table for the orders on the waiting list has to be added and the handler of the incoming products has to check whether to start picking according to a waiting order.

M-10: Adding a new user type will cause small changes in the content of the database, because the user types are stored in a database table of their own. The user data includes information about the user type of each user. Of course also the enumeration type of the different user types has to be expanded. Every function of the system is only active for the users that have the right to perform that action and this will be checked in the database so only the rights database table needs to be updated.

M-11: To add hour reporting to the system one needs to define a new message handler for all the functions needed for dealing with the hour reports. This should not be a problem because it does not require changes in the other message handlers. New displays should also be easy to add to the old user interfaces. The greatest change will be the modification of the message queue so that it will know how to handle the messages concerning a new function.

M-12: To be able to categorize the functions of the system also according to which of them are allowed to be performed through a remote connection, an additional database table has to be defined to store the rights of execution. All the displays of the remote connections have to be changed to check whether the function is allowed or not.

M-13: The changes required by the currency change are of course only temporary. During the changing period of the currency all the displays dealing with money need to be modified temporarily to show both the currencies. The invoices have to be done showing both the currencies so the message handlers dealing with them have to be changed too. When receiving a deposit one has to check in which currency it is done. This will cause changes in the displays, the handlers, the interface with the ERP, and the XML format.

M-14: There are two different ways to add the registration to the reception of the products. One option is to simply modify the reception handler to register the product as well, if it is not yet registered. This one is not so good because then the system will have the same code in two different procedures. Considering the reusability of the system the better option is to make the reception handler call the registration handler. This will however modify the systems call graph and the top level of the architecture. The other message handlers do not call each other but only handle the messages from the queue.

M-15: If the number of clients exceed the capacity of the client manager, the best way to make the system more scalable is to add another client manager. This will very likely mean that the message handlers are not efficient enough, and probably the handlers for the most often needed functions should also be doubled. The biggest change will be in the message queue, because it has to be modified so that the client managers can use it parallel.

4.6.2 Assessing the reusability scenarios

Both the scenarios seem to be almost completely reusable, because they were implemented to represent typical products of the domain of DoneWMS. The message queue will be reused with no changes in both the scenarios, because it has been implemented for such use. The components are compatible with both the operating systems because DoneWMS was planned for both of them.

R-1: This system does not need all the components of DoneWMS. The clients require minor changes because some of the functions need to be removed from the displays. That does not affect the handlers, because each function has a handler of its own, so some of the handlers will just be left out. The handlers used by the system can be reused almost completely because no special features were added.

R-2: This system will use all the components available for DoneWMS. The client manager will not be completely reusable, and there will be changes similar to the ones in scenario M-15. The displays need not be modified, because all the functions are used.

It is not enough just to list the components that can be reused to make the assessment. We should also estimate how many lines of code the reusable components contain and how many lines of code the whole system contains to be able to count the percentage of reused lines of code. The estimate of the number of lines would be very rough due to the lack of our knowledge of the system, so we will discard it in the assessment.

A better way to define the magnitude of the changes would be to estimate the complexity of the components instead of the number of lines of code. This is, however, not possible with scenario-based assessment. Metrics-based assessment could be used for assessing the fan-in and fan-out of the components and defining the complexity by them.

The assessment of reusability will be stalled due to the lack of information about the system. With SAAM we cannot define whether the limit of 80 % lines of reused code is achieved.

4.7 Revealing the scenario interaction

The results of the modifiability scenario assessments are summarized in Table 4.6. 'X' stands for major changes and 'x' for minor changes in the part of the system mentioned on the top row of the table. M-7 is marked with '?' because it is not certain if any changes occurred.

The first thing that attracts attention in the summary of the scenarios, is that scenarios M-3 and M-4 cause changes in exactly the same components. However, this is not peculiar, because the scenarios were developed of the same category, so they both represent the same class of scenarios and actually should cause changes in exact same components.

Table 4.6: Summary of the change scenarios.

scenario	handler (s)	ERP interface	XML format	data-base	displays	MQ	client manager	architecture description
M-1	X		x					
M-3	X	X		X	X			
M-4	X	X		X	X			
M-6	X	X		x				
M-7	?			?		?	?	
M-8	X							X
M-9	X			X				
M-10				x				
M-11	X					x		
M-12				X	X			
M-13	X	X	X		X			
M-14	X							X
M-15						X	X	

There is not much interaction within the scenarios. Only M-8 and M-14 are interacting in the handlers and they both also cause changes at the highest level of the architecture description. They both concern changing the call graph of the system. In both the scenarios the handlers need to call something else but the JDBC, which is not normal for the handlers.

The rest of the scenarios require changes in some of the components but none of them are interacting in the actual meaning of the definition of scenario interaction. Even if they cause changes in the same component the other components they will change are different.

4.8 Overall assessment

At the beginning of the assessment we decided that the system is easily modifiable if the most probable changes can be done by changing only two components of the system. This was supposed to be true for the most probable 50% of the scenarios i.e. M-1, M-2, M-5, M-8, M-10, M-11 and M-15. This was achieved fairly well on DoneWMS. Only one of these scenarios (M-2) requires changes in four components. On the other hand one of the scenarios was direct and one required changes in only one component, so we can say that the modifiability of DoneWMS reaches the level we set for good modifiability in this assessment. The next question is, whether this limit is realistic for the modifiability of a WMS.

Change category C was the only category of which more than one scenarios were developed. Two of these scenarios (M-3 and M-4) caused the same changes to the system, which means they were of the same class of scenarios. However the third scenario of this category was direct. The scenario categories were supposed to be divided into subcategories, so that each

of the subcategories represents only one class of scenarios. DoneWMS was apparently a little too complex to be divided into categories by this method, or twelve categories were simply too few to assessing the whole system even if it appeared to be sufficient at the beginning. This might have an effect on the accuracy of the whole assessment, but it is difficult to know whether the other categories are divided into small enough subcategories or not, without doing the whole assessment again from the beginning.

The interactions of the scenarios seems good. The fact that the two scenarios interacting were alike, denotes high cohesion. In this case it means that similar changes in the systems call graph can be treated in a coherent way.

If there had been interactions in scenarios that were not alike, problems of the architecture could have been revealed. The fact that problems could not be revealed does not necessarily mean that the architecture is perfect. The problem may also be in the assessment. For instance in this case the scenario categories were not divided until the end or the architecture was not described detailed enough. For the latter, there was not much to be done in this case study, and the former could have been a consequence of the latter.

The limit set for good reusability at the beginning of the assessment was 80% of reused code lines in an application that represents the domain of DoneWMS. We did not carry out assessment of reusability until the end but a rough estimate of the scenarios we did develop is that 80% could very well be achieved. This estimate is not very reliable, because it was done according to only two scenarios. To make the estimate more reliable a bigger number of scenarios should be developed. This would, however, require a more careful analysis of the variability points of DoneWMS.

To summarize the results of the assessment, the modifiability of the system seems to be good since all the change scenarios were realizable by slight modifications and the limit set for modifiability was almost achieved. We cannot say much about reusability and nothing at all about the learnability of DoneWMS. This is a disadvantage of the assessment and should be considered carefully.

5 Analysis of the assessment

The assessment was not as easy to carry out as it seems when reading through the case study and it is important to analyze why it is so. The fact that the exact knowledge of the system was not very accurate can be left out of the observation and we can concentrate on the weak points of the assessment method and analyze each step of the process one by one to analyze which parts of the scenario-based assessment could be approved.

5.1 Choosing the quality attributes

At the beginning of the assessment, performance was already left out from the assessment, because it was considered not possible to be assessed by our knowledge of the architecture. Learnability is actually not a very architectural quality attribute either, but it was kept in the assessment until the assessment could not be carried further with it.

Section 2.3. claims that questioning techniques of architectural assessment can assess any given quality attribute. How come carrying out these two assessments was still so difficult for some of the quality attributes? We managed to assess modifiability, which is a highly architectural quality. The assessment of reusability was also carried out until almost the end and reusability is an architectural quality too. The quality attributes that there were problems with were the ones that are not only architectural. This can be interpreted that even if the assessment of all the quality attributes is possible with questioning techniques, it is much easier for architectural qualities, when performed as architectural assessment. Questioning techniques can also be used for other assessments besides the architectural assessment. Before the assessment, the assessment team should consider which of the qualities have to be

assessed already at the beginning of the development process and which can be done later on. For example, the displays of the system can be developed and assessed by a usability specialist at any stage of the development process.

5.2 Choosing the assessment method

In this case study the assessment method was chosen to be SAAM, because the architecture description of DoneWMS provides the description of the functionality of the architectural components. ATAM does not require these descriptions so it was natural to choose SAAM to make use of them.

The description of the functionality was not very accurate and it could not be used very effectively, so maybe ATAM would have been a better alternative, because it would not have required so much information. More detailed instructions on which method to choose would have been useful at the beginning, especially because of the unexperienced assessment team.

5.3 Creating the scenarios

The most difficult part of the assessment was creating the scenarios. That was not only due to the lack of material about DoneWMS but also because the assessment team did not have prior experience of warehouse management systems or assessing them or any other architectures. The assessment was done only according to theoretical knowledge. An approximate assessment was nevertheless possible.

Dividing the system into categories facilitated finding the scenarios and it also helped in making sure that the set of scenarios was closer to a complete set than without the categories. Of course the set would have been nearer to a complete set, if the assessment team had been bigger and had had better knowledge of DoneWMS.

Even if dividing the system into categories was helpful, it required a lot of planning to choose the division of the categories, because in such an ambiguous system there are always many different ways of dividing the categories. If the categories do not cover the whole system, choosing the scenarios according to them does not make the scenario set any better than a randomly chosen set. The greatest difficulty was to figure out whether the categories cover the whole system. For example the scenario M-13 did not fall in any of the categories even if an attempt to divide them in order to cover the whole system was made. There was a need for a method for dividing the categories in a systematic way so that one could ensure that the whole system will be covered or a method for verifying that the categories divided by common sense cover the whole system.

Another problem in dividing the categories was to figure out when the subcategory presents one class of scenarios and when it still needs to be divided into subcategories. As was revealed at the end of the assessment, the category C did not include just one class of scenarios.

Of course this could have been verified for each of the categories individually by defining more scenarios for of each of them and iterating the division as long as the scenarios were all of the same category. When assessing an ambiguous system like this, it would require too much work and it would become too expensive, so usually it is impossible.

The intention in Section 4.4. was to create scenarios that will cause more changes to the architecture with the knowledge gained in Section 4.3.. This was not very successful, because the internal structure of the components was not revealed. The architect of the system would have been very useful in this section. Of course the architect has to be willing to find the problematic scenarios to be useful for the assessment.

Probably the greatest problem faced with when dividing the scenarios was that the assessment team was not big enough, so it lacked the different points of view that larger teams have. Unfortunately this is also a lack of the assessment method and not only of this case study. The assessment needs a team. One person is not enough.

5.4 Classifying the scenarios

Classifying the scenarios was found not to be too easy either even if it sounded easier than creating the scenarios. The distinction between direct and indirect scenarios was not too clear. In the end almost all the scenarios were indirect because they required some kind of changes even if some of the changes were so small that they could have been recognized as direct scenarios. If the assessment of the work needed for performing the changes is left out from the assessment, as is the case in this case study, there is also no way to define an exact limit of the direct and indirect scenarios, which makes the assessment not so accurate.

To be sure of the division between the direct and indirect scenarios one has to do some outlines of the scenario assessment for all the scenarios first. Maybe the step ”Performing scenario assessments” could come before the step ”Classifying the scenarios”, because the direct scenarios have to be assessed to figure out they are direct. This can be very different even if the architect is involved in the assessment. With good knowledge of the architecture one can say directly whether the scenario is direct or not.

5.5 Scenario assessments

As mentioned already, in this assessment the scenario assessments did not contain the assessment of the work needed for performing the changes to the system. This would have been more important for the assessment results if the assessment had been comparative. It was possible to carry out the absolute assessment without assessing the work as well.

Making the assessments of the work needed for the changes could have also been a way to measure the reusability of the system and that is why they should have been done in

this assessment too. Unfortunately it was not possible due to the superficial architecture description.

The assessments of the work needed for making the changes can also be used for classifying the scenarios so probably it should be a part to be included in all the assessments, also the absolute ones. This also pleads that the places of the step ”Performing the scenario assessments” and the step ”Classifying the scenarios” could be changed in the assessment method.

The scenario assessments were superficial overall, but the most important parts of the assessment were performed successfully in spite of this. When performed to a customer, the assessments have to be more in depth and more concentrated on scenario assessments to improve the assessment results.

5.6 Scenario interaction

Assessing the scenario interaction must be a good way to find the parts of the architecture that, for example, deal with more than one function or are too complex in some other way. To prove that the system does not have these parts too complex, the scenarios do not provide much support. This is, of course, natural with methods whose aim is to point out problems. It is not likely that this can be changed by improving the way in which the interaction is assessed. Instead an other kind of assessment should be added to the end to verify the architecture really suits its usage in case problems could not be found.

5.7 Overall assessment

In this case study the overall assessment was on one hand positive, because the only things said about the architecture were positive. The only measurement carried out until the end almost satisfied the limit set at the beginning and the interaction revealed denoted cohesion instead of complexity.

On the other hand the feedback of the assessment was not positive, because problems could not be discovered. This is not useful for further processing. If an assessment can not reveal problems, it would be good to be able to ensure that there are no problems. However, with scenario-based assessment this is not possible.

5.8 Assessment team

The composition of the assessment team was one of the biggest problems of the assessment. The team definitely needs to consist of more than just one person and some of the persons

involved need to know the system very well. Some people with experience of assessments would also be a good help, but this problem can be overcome by careful preparations.

Making the assessment team bigger does not necessarily need to make the assessment much more expensive because of multiplying the working hours. The larger and more initialized to the system the team is, the more intensive the assessment will be. Time will not be wasted on searching for the problems but for assessing them. Of course not all of the team need to take part in all the assessment steps either. For instance the architect can be left out of the "Introduction of the architecture".

5.9 Improvements

To improve the results of the assessment, other assessment methods could have been included. Most of the other assessment methods would not have suited for this case study alone, because for instance a simulation of the system would have been totally impossible to accomplish with such little information about the system, but a combination of two methods could have been worth trying.

If there had been more information available about the system, the assessment could have concentrated also on the quantitative assessment of the quality attributes. For instance performance would not have been left out of the assessment and the performance issues brought up by the scenario-based assessment could have been assessed more carefully with the metrics.

Throughout the assessment, not only in assessment of performance, the combination of scenario-based and metrics-based assessment would probably have given better results. If there is time for the assessment a combination of two assessment methods is a good idea.

In the assessment of reusability a more clear approach towards the product-line architecture would have been useful. Using a domain and variability analysis as the groundwork of the assessment will improve the scenarios and the results of the whole assessment. The assessment of product-line architectures does not stand alone but it also needs an analysis of the domain and variability of the product line. Without these the assessment will be more like assessing a single architecture.

The familiarity with the system, as has been pointed out already several times, is important for the assessment and improves the results considerably so it needs to be stressed once again. If one wants to invest on the assessment, taking time from the architect of the system is the most important thing, after good groundwork of course. A person with good knowledge of the system is totally on another level, when creating the scenarios and assessing them. Of course the outsiders point of view should never be discarded.

If there had been a chance to have the system's architect in the assessment the division of the work could have been that the outsiders had created the scenarios or at least done the first round of scenario development alone and the architect had participated mainly in the steps

”Classifying the scenarios” and ”Performing the scenario assessments”, because those were the parts most affected by the lack of the knowledge of the architecture and the scenario development will benefit of the new points of view that the outsiders have.

On the whole the assessment did not seem very successful due to the lack of concrete assessment results, but the benefits of this assessment lay elsewhere. The assessment method was reviewed thoroughly and some points about its usage were considered. In this case study the expectations were not too high because the architecture was not to be assessed thoroughly. The assessment, taken more so as an example, was sufficient to raise questions about the assessment method and its usage.

6 Conclusions

Because the software architecture is the product of the first design phase in the development process of the software system, the assessment of the architecture is a powerful tool. This is mainly on account of the possibility to do it early in the development process and its affordability. One should have realistic expectations with the assessment though. One should not expect that the assessment will reveal all that will take place in the development process but to realize that with limited knowledge of the system the gained information will also be limited.

Architectural assessment is especially useful in getting to understand the architecture. Architecture itself is a communication tool of the development team and the main benefits of the assessment are its communicational uses.

Scenario-based assessment is a technique that can easily be applied in projects that are only at the beginning and cannot provide very accurate data of the system. If the architecture description is superficial the results will not be very accurate either, so the biggest benefit from scenario-based assessment is that the feedback is gained at an early stage of the development. To make the results more accurate scenario-based assessment could be combined with another assessment method, for example the metrics-based assessment, but this cannot be done in at such an early stage of the development process unlike the scenario-based assessment alone.

The software architecture analysis method is not as detailed as the architecture tradeoff analysis method. Thus it is easier to apply to an architecture that is not very accurately described.

The most important step of SAAM assessment is the development of scenarios. If it is not done carefully the assessment will not be successful. It is difficult to ensure that the set of

scenarios is complete. Dividing the possible scenarios into categories is helpful but does not ensure the completeness of the scenario set. A detailed method for dividing the scenarios would be a helpful tool to be used with SAAM.

If the knowledge of the architecture is not too good, the most difficult step of SAAM assessment can be ”the scenario assessments”. In this case the benefits of the assessment will be the understanding of the system. If it is possible to have a person that already knows the architecture participating in the assessment, the assessment can concentrate more on finding the weak points of the system. In this case the assessment team should consider which part of the assessment the systems architect should participate in. The knowledge of the system is most needed in ”the scenario assessments” so the architect should participate in it. On the other hand new points of view are good for ”the scenario development” so the architect probably should not take part in it. Overall, the participation of the architect will most likely improve the assessment results considerably.

Moreover, it is not enough just to have an outsiders’ point of view and the systems architect in the assessment team. The assessment team should also have more people in it to provide different points of view. Of course the price of the assessment has to be taken into account before making the assessment team very large but it does not always mean that the assessment will be more expensive if there are more people involved in it. And usually it is beneficial to invest a little bit more if the assessment is decided to be done.

After all, the architectural assessment is an inexpensive way to avoid huge changes in the development process. The organization that is involved in the development process and the assessment should be used before the problems occur i.e. as early as possible. There is no use assessing a complete product that is already almost impossible to change.

Bibliography

[Bass et al., 1998] Len Bass, Paul Clements and Rick Kazman: *Software Architecture in Practice*. Addison Wesley (1998).

[Bosch, 2000] Jan Bosch: *Design and use of Software Architectures: Adopting and evolving a product-line approach*. Addison Wesley (2000).

[Clements et al., 2001] Paul Clements, Rick Kazman and Mark Klein: *Evaluating Software Architectures: Methods and Case Studies*. Addison Wesley (2001).

[Dobrica et al., 2000] Liliana Dobrica and Eila Niemelä: *A strategy for analysing product line software architectures*. Espoo 2000, Technical Research Center of Finland, VTT Publications 427.

[Jakovlev et al., 2001] Igor Jakovlev, Edgar Popovici, Cozma Hrincescu and Mika Ylikangas: *DoneWMS Warehouse Management System*, Software architecture document, v 2.0, Technical report, Done Logistics (2001).

[Kazman et al., 2000] Rick Kazman, Mark Klein and Paul Clements: *ATAM: Method for Architecture Evaluation*. Technical report, CMU/SEI-2000-TR-004, ESC-TR-2000-04

[Ylikangas, 2001] Mika Ylikangas: *DoneWMS Warehouse Management System*, System specification, v 2.0, Technical report, Done Logistics (2001).