Modifiability through Architecture Analysis

Nico Lassing[1] , PerOlof Bengtsson[2], Jan Bosch[3], Daan Rijsenbrij[1] and Hans van Vliet[1]

[1]Division of Mathematics and Computer Science
Faculty of Sciences, Vrije Universiteit
Amsterdam, The Netherlands

[2]Department of Software Engineering and Computer Science
Blekinge Institute of Technology
Ronneby, Sweden

[3]Department of Mathematics and Computer Science
University of Groningen
Groningen, The Netherlands

Abstract

Software evolves. One of the challenges that organizations are facing is how to reduce the costs associated with these adaptations. To address this issue, software architecture is seen as an important tool. A system's software architecture is generally regarded to have a large influence on the effort required to adapt a system [1]. Therefore, analysis of the decisions made at the architectural level is important to achieve modifiability and reduce the costs of maintenance. The area addressing this is *software architecture analysis of modifiability*. We have developed a method for this type of analysis, which is presented in this paper.

1 Introduction

Many organizations today operate in a world in which change is ubiquitous. Business developments and new technology force them to adapt their information systems regularly. Software architecture is seen as an important tool to reduce the costs associated with these adaptations. A system's software architecture is the first artifact in the development process and captures the very first design decisions for that system [1]. These decisions not only concern internal aspects of the system, i.e. its decomposition in components and the dependencies between these components, but they also concern the system's environment, i.e. dependencies among systems and their relationship to the technical infrastructure. Decisions on both levels have a large influence on the effort that is required to modify the system once it is built. Therefore, modifiability should not only be considered when making these decisions, but it is also important to assess whether the chosen software architecture results in the required modifiability. The area addressing this second aspect is software architecture analysis (SAA) of modifiability.

We have defined a generic method for software architecture analysis of modifiability. This method is based on change scenarios, meaning that we capture likely, concrete events that may occur in the life cycle of the system and explore their effect on the system by studying the software architecture. This allows us to make predictions about the modifibility of the system before it has been built.

The method is a generalization of the work of the authors independently (see for instance [7] and [2]) and was succesfully applied in various areas, such as telecommunications, logistics and business information systems. This paper gives an overview of the method.

2 The generic method for SAA of modifiability

The generic method for modifiability assessment of software architecture that we advocate is based on the Software Architecture Analysis Method

(SAAM) [5]. They both use scenarios to analyze the software architecture of a system. The main difference between the two is that we focus exclusively on modifiability analysis and explicitly distinguish various goals in this area, namely risk assessment, maintenance prediction and architecture comparison.

We define modifiability in this context as the ease with which a system can be adapted to changes in the functional specification, in the environment, or in the requirements. We explicitly exclude the correction of implementation errors and changes in the quality requirements of the system. Changes in the quality requirements are left aside, because the assessment of their effect belongs to other disciplines. Changes in the required processing capacity of a system, for example, relate to scalability, which is a field of study in itself (see [10]).

Our method consists of five steps:

1. Set goal: determine the aim of the analysis
2. Describe software architecture: give a description of the relevant parts of the software architecture
3. Elicit change scenarios: find the set of relevant change scenarios
4. Evaluate change scenarios: determine the effect of the set of change scenarios
5. Interpret the results: draw conclusions from the analysis results

Although this list suggests otherwise, these steps are not always performed in the indicated sequence; it is often necessary to iterate over them. In this section, we give a overview of the various steps. A more elaborate discussion of the full method is given in [3].

2.1 Goal setting

The first step to take in software architecture analysis of modifiability is to set the analysis goal. The goal determines the type of results that will be delivered by the analysis. In addition, the goal influences the choice of techniques to be used in subsequent steps. Different goals ask for different techniques. With respect to modifibility, the following goals can be pursued: (1) risk assessment, (2) maintenance prediction and (3) software architecture comparison. The aim of risk assessment is to find types of changes for which the system is inflexible, i.e. scenarios which are particularly difficult to accomplish. We call this risk assessment because these changes may be a threat to the evolution of the system.

On the other hand, when performing maintenance cost prediction the aim is to estimate the cost of (adaptive) maintenance effort for the system in a given period. In a very general sense, we then use a maintenance cost function $C_{average}$ (the average cost per change scenario) of the form

$$C_{average} = \frac{\sum_{i=1}^{n} C(\mathit{change}_i) \times p(\mathit{change}_i)}{n}$$

where $C(\mathit{change}_i)$ denotes the effort or cost required to realize the i-th change scenario, and $p(\mathit{change}_i)$ denotes the probability this scenario will occur.

Finally, if the goal is software architecture comparison, we are comparing two or more candidate software architectures to find the most appropriate one. The difference between comparison and the two aforementioned goals is that in comparison we make relative statements about a number of candidate software architectures, while when pursuing the other goals we make absolute statements about a single candidate architecture. In comparison, we are most interested in those change scenarios that have a distinct effect on the various candidates.

2.2 Software architecture description

After the goal of the analysis is set, the next step is to create a description of the software architecture. This description should enable us to do architecture level impact analysis for the set of change scenarios. The information acquired in this step is recorded in a number of architectural views.

An architectural view is a model of the software architecture. It is commonly understood that the software architecture cannot be captured in a single model. Therefore, a system's software architecture is usually recorded in a number of views. Each architectural view provides a particular perspective on the software architecture capturing a set of design decisions, such as the run-time perspective or the deployment perspective. A collection of such views is called a *view model*. A number of authors have introduced view models, which consist of a coherent set of views. The most important of these are the 4+1 view model by Kruchten ([6]) and the four architectures by Soni, *et al.* ([9] and [4]).

To perform architecture level impact analysis we need those views that capture structural aspects of the system, i.e. the system's decomposition in components, the relationships between these components and the relation of the system to its environment. This information enables us to determine which components have to be adapted, including their ripple effects.

In the scenario evaluation step, we may discover that essential information is missing. In that case, we have to revisit this step and elaborate the description as needed.

2.3 Change scenario elicitation

One of the most important steps in modifiability analysis is the elicitation of a set of change scenarios. This set of change scenarios captures the events that stakeholders expect to occur in the future of the system. The main technique to elicit this set is to interview stakeholders, because they are in the best position to predict what may happen in the operational life of the system. Stakeholders are people like the owner of the system, future users, developers and maintenance staff. It is important to involve all of them in the analysis process, because each of the stakeholders has a different perspective on the evolution of the system and all these aspects are important.

The set of change scenarios that we acquire should support the goal that we have set for the analysis. If the goal of the analysis is risk assessment the elicitation process should be focused on those change scenarios that are particularly difficult to implement. This means that we should stimulate the stakeholders to come up with change scenarios that are complex.

If the goal of the analysis is maintenance prediction, the scenario elicitation process should be aimed at finding the set of most likely change scenarios. In addition to a description of the change scenarios, we also need to come to a measure to indicate their probability. This measure can also be determined in the interviews with the stakeholders of the system, because they are probably in the best position to determine a scenario's likelihood.

If comparison is the aim of our analysis, the set of change scenarios should highlight the differences between the two candidate architectures. Change scenarios that have the same effect on all candidates provide no real value to the analysis, only those that discriminate between them. Such discriminating scenarios may be deduced from the characteristics of the candidate architectures.

A possible limitation of our approach is that we rely on stakeholders to come up with change scenarios. It is not unlikely that some scenarios may be overlooked. This may be addressed by using a more structural approach to scenario elicitation, in which a theory of change scenarios is used to guide the scenario elicitation process. This has proven to be useful in our case studies. For a more elaborate discussion of this approach see [3] and [8].

2.4 Change scenario evaluation

After eliciting a set of change scenarios, we determine their effect on the software architecture. This means that we perform architecture level impact analysis for each of the scenarios individually. To do so, we first have to determine the components of the system and components of other systems that have to be adapted to implement the change scenario. This task is typically performed in collaboration with members of the development team.

After we have determined the components that should be adapted, the results are expressed in such a way that it supports the goal of the analysis. For risk assessment this means that the results should be expressed in such a way that it helps us to get insight into the complexity of the required changes. In [7] a number of factors is mentioned that we found to influence the complexity of changes for business information systems. These factors include the question whether the software architecture needs adaptation, the involvement of several system owners, and the introduction of multiple versions of the same component. These factors are not fully comparable, but they help us gain insight into the complexity of the required changes.

On the other hand, when the aim of the analysis is maintenance prediction, the results should be expressed in such a way that we may determine the effort that is required to implement the change scenario. Possible measures include the number of lines of code that are affected, and the aggregate size of the components that are affected. The choice for a specific set of measures depends on the prediction model used.

For architecture comparison, the results of the evaluation should be expressed in such a way that it enable us to compare the candidates. Various strategies can be employed. For instance, we may determine for each change scenario the candidate architecture that supports it best, or if there are no differences. Alternatively, we can rank the candidate architectures for each change scenario depending on their support for the scenario. Or otherwise, we can determine the effect of the scenarios individually on each candidate architecture and express this effect using some scale, e.g. a five level scale (++, +, +/-, -, and --) or the number of lines of code affected. In the next step these results are then interpreted to come to an overall comparison.

2.5 Interpretation

After we have determined the effect of the change scenarios, we can interpret these results to come to a conclusion about the system under analysis. The way the results are interpreted is again dependent on the goal of the analysis. However for each goal it is important that the likelihood of the scenarios is considered in this selection process.

If the goal of the analysis is risk assessment the results of the scenario evaluation are investigated to determine which change scenarios pose risks, i.e. for which scenarios the product of probability and costs is too high. The criteria for determining which values are still acceptable should be based on managerial decisions by the owner of the system. When risks are found, various risk mitigation strategies are possible: avoidance (take measures to avoid that the scenario will occur or take action to limit their effect, for instance, by use of tools), transfer (choose another software architecture) and acceptance (accept the risks).

In maintenance prediction, the aim of this step is to come to an estimate of the amount of effort that is required for maintenance activities in the coming period. To this end an estimation function is used as given in section 2.1. The estimate can then be compared with the requirements that were formulated for the system.

If the goal of the analysis is architecture comparison, we compare the results of the evaluation of the two sets of scenarios and choose the most appropriate candidate architecture.

3 Summary

This paper gives an overview of our method for software architecture analysis of modifiability. This method consists of five steps in which we employ scenarios, capturing concrete and likely changes, to gain insight into the modifiability of a system. The first step is to set the goal of the analysis. We discussed the three goals that we distinguish for modifiability analysis: risk assessment, maintenance prediction and architecture comparison. The second step is to create a description of the software architecture under analysis. This description serves as a basis for the rest of the analysis process. The third step is to interview stakeholders to elicit a set of change scenarios. The effect of these scenarios is determined in the next step, the result of which is expressed using an appropriate measure. Subsequently, these results are interpreted to reach a conclusion about the modifiability of the system. This method has succesfully been applied in a variety of settings, such as logistics, telecommunications and business information systems.

Acknowledgements

We are very grateful to Cap Gemini Netherlands for their financial support of this research.

References

[1] L. Bass, P. Clements, and R. Kazman. *Software Architecture in Practice*. Addison-Wesley, Reading, Massachusetts, 1998.

[2] P. Bengtsson and J. Bosch. Architecture level prediction of software maintenance. In *Proceedings of the 3rd European Conference on Software Maintenance and Reengineering*, pages 139–147, March 1999.

[3] P. O. Bengtsson, N. Lassing, J. Bosch, and H. van Vliet. Analyzing software architectures for modifiability. Technical Report HK-R-RES–00/11-SE, Högskolan Karlskrona/Ronneby, 2000.

[4] C. Hofmeister, R. Nord, and D. Soni. *Applied software architecture*. Addison-Wesley, Reading, Massachusetts, 1999.

[5] R. Kazman, G. Abowd, L. Bass, and P. Clements. Scenario-Based analysis of software architecture. *IEEE Software*, 13(6):47–56, 1996.

[6] P. B. Kruchten. The 4+1 view model of architecture. *IEEE Software*, 12(6):42–50, 1995.

[7] N. Lassing, D. Rijsenbrij, and H. van Vliet. Towards a broader view on software architecture analysis of flexibility. In *Proceedings of the 6th Asia-Pacific Software Engineering Conference '99 (APSEC'99)*, pages 238–245, 1999.

[8] N. Lassing, D. Rijsenbrij, and H. van Vliet. Why I come to a better set of change scenarios than you. Technical report, Vrije Universiteit, Amsterdam, 2000.

[9] D. Soni, R. L. Nord, and C. Hofmeister. Software architecture in industrial applications. In

R. Jeffrey and D. Notkin, editors, *Proceedings of the 17th International Conference on Software Engineering*, pages 196–207, New York, 1995. ACM Press.

[10] M. van Steen, S. van der Zijden, and H. Sips. Software engineering for scalable distributed applications. In *Proceedings 22nd International Computer Software and Applications Conference (CompSac)*, 1998.