Automating Software Architecture Assessment

Jilles van Gurp & Jan Bosch[1]

University of Karlskrona/Ronneby

Abstract. In this paper we present SAABNet (Software Architecture Assessment Belief network), an approach to automating the process of performing software architecture assessment. We have found that SAABNet is especially useful early in the development process when measurable assets are scarce. In this stage of development software architects have little more than their own experience to rely on. SAABNet tries to capture this experience and use it to help architects perform assessments.

Introduction

Traditionally the software development is organized into different phases (requirements, design, implementation, testing and maintenance). The phases usually occur in a linear fashion (the waterfall model). The phases of this model are often repeated in an iterative fashion. This is especially true for the development of OO systems.

At any phase in the development process, the process can shift back to an earlier phase. If, for instance, during testing a design flaw is discovered, the design phase and consequently also the phases after that, need to be repeated. These types of setbacks in the software development process can be costly, especially if radical changes in the earlier phases (triggering even more radical changes in consequent phases) are needed. We have found that non-functional requirements or quality requirements often cause these type of setbacks. The reason for this is that testing whether the product meets the quality requirements generally does not take place until the testing phase [1].

To assess whether a system meets certain quality requirements, several assessment techniques can be used. Most of these techniques are quantitative in nature. I.e. they measure properties of the system. Quantitative assessment techniques are not very well suited for use early in the development process because incomplete products like design documents and requirement specifications do not provide enough quantifiable information to perform the assessments. Instead developers resort to qualitative assessment techniques. A frequently used technique, for instance, is the peer review where design and or requirement specification documents are reviewed by a group of experts. Though these techniques are very useful in finding the weak spots in a system, many flaws go

1. [Jilles.van.Gurp|Jan.Bosch]@ipd.hk-r.se, University of Karlskrona/Ronneby, Department of Software Engineering and Computer Science, Soft Center, S-372 25 Ronneby, Sweden, http://www.ipd.hk-r.se/[jvg|bosch]

unnoticed until the system is fully implemented. Fixing the architecture in a later stage can be very expensive because the system gets more complex as the development process is progressing.

Qualitative assessment techniques, like the peer review, rely on qualitative knowledge. This knowledge resides mostly in the heads of developers and may consist of solutions for certain types of problems (patterns [2][3]), statistical knowledge (60% of the total system cost is spent on maintenance), likely causes for certain types of problems ("our choice for the broker architecture explains weak performance"), aesthetics ("this architecture may work but it just doesn't feel right"), etc. A problem is that this type of knowledge is inexplicit and very hard to document. Consequently, qualitative knowledge is highly fragmented and largely undocumented in most organizations. There are only a handful known ways to deal with this problem:

- Assign experienced designers to a project. Experienced designers have a lot of knowledge about how to engineer systems. Experienced designers are scarce, though, and when an experienced designer resigns from the organization he was working for, his knowledge will be lost for the organization.
- Knowledge engineering. Here organizations try to capture the knowledge they have in documents. This method is especially popular in large organizations since they have to deal with the problem of getting the right information at the right persons in the organization. A major obstacle is that it is very hard to capture qualitative knowledge as discussed above.
- Artificial Intelligence (AI). In this approach qualitative knowledge is used to build intelligent tools that can assist personnel in doing their jobs. Generally, such tools can't replace experts but they may help to do their work faster. Because of this less experts can work more efficiently.

We followed the latter approach and used the Bayesian Belief Network (BBN) technique to create SAABNet (Software Architecture Assessment Belief Network). SAABNet enables us to feed information about the characteristics of an architecture to the program. Based on this information, the system is able to give feedback about other system characteristics. The SAABNet BBN consists of variables that represent abstract quality variables such as can be found in McCall's quality factor model [4] (i.e. maintainability, flexibility, etc.) but also less abstract variables from the domain of software architectures like for instance inheritance depth and programming language. The variables are organized in such a way that abstract variables decompose into less abstract variables.

We have published a more extensive study of SAABNet in [6]. In that paper we provide a description of the specification of SAABNet as well as a validation of SAABNet. In this paper we briefly summarize the essentials of SAABNet and demonstrate its use. We were not the first to apply belief networks to software engineering. In [12] and [13], BBNs are used to assess system dependability and other quality attributes. Contrary to our work, their work focuses on dependability and safety aspects of software systems.

Bayesian Belief Networks

A Bayesian Belief Network is a directed acyclic graph. The nodes in the graph represent probability variables and the arrows represent conditional dependencies (not causal relations!). A conditional dependency of variable C on A and B in the example in figure 1 means that if the probabilities for A and B are known, the probability for C is known. If two nodes are not directly connected by an arrow, this means they are independent given the nodes in between (D is conditionally independent of A). Each node can contain a number of states. A conditional probability is associated with each of these states for each combination of states of their direct predecessors (see figure 1 for an example).

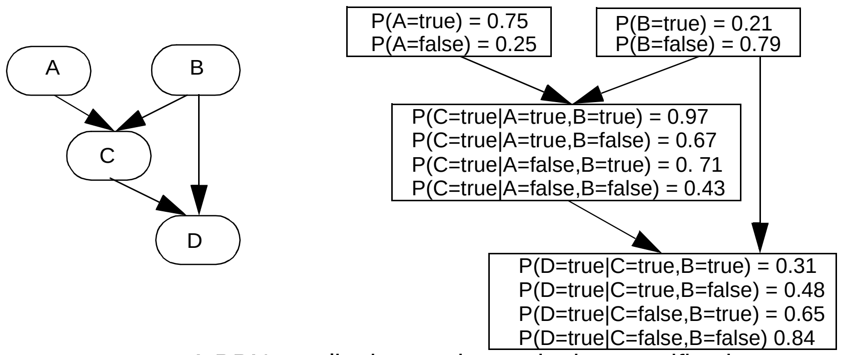
A
B
C
D
P(A=true) = 0.75
P(A=false) = 0.25
P(B=true) = 0.21
P(B=false) = 0.79
P(C=true|A=true,B=true) = 0.97
P(C=true|A=true,B=false) = 0.67
P(C=true|A=false,B=true) = 0. 71
P(C=true|A=false,B=false) = 0.43
P(D=true|C=true,B=true) = 0.31
P(D=true|C=true,B=false) = 0.48
P(D=true|C=false,B=true) = 0.65
P(D=true|C=false,B=false) 0.84

Figure 1 A BBN: qualitative and quantitative specification.

A BBN consists of both a qualitative and a quantitative specification. The qualitative specification is the graph of all the nodes. The quantitative specification is the collection of all conditional chances associated with the states in each node. In figure 1 a qualitative specification is given and a quantitative specification is given in figure 1.

By using a sophisticated algorithm, the a priori probabilities (i.e. the probability for a variable given the probabilities of all the other variables in the network) for all of the variables in the network can be calculated using the conditional probabilities in the quantitative. This would take exponential amounts of processing power using conventional mathematical solutions (Bayes' theorem) since it's a NP complete problem. A BBN can be used by entering evidence (i.e. setting probabilities of variables to a certain value). The a priori probabilities for the states of the other variables are then recalculated. How this is done is beyond the scope of this paper. For an introduction to BBNs we refer to [5].

SAABNet

SAABNet was developed as a proof of concept to verify whether a technique such as bayesian belief networks would be of use to assist in software architectures. The main reasons that caused us to believe that such a technique could be of use were:

- Architecture assessment in early phases of the development of a system is difficult due to a lack of measurable assets. BBNs don't need complete information to deliver usable results.

- Qualitative architecture assessment is generally done by architecture experts who have to deal with a wide range of inherently uncertain knowledge. BBNs are able to work with uncertain knowledge.
- Important design decisions are made early in the development process and errors made in this stage are hard to correct later on. BBNs can be applied earlier than conventional quantitative assessment techniques because they don't require complete and certain information.

We believe that a non-metrics based approach is the only feasible way of doing architecture assessment early in the development of an architecture. All metrics-based approaches are essentially useless early on since there is a lack of measurable products and the value of the metrics that are available is limited. There is however a wealth of qualitative knowledge that can and should be used when assessing architectures.

While we cannot claim to be expert software designers we do have general insights in what constitutes good design [7], the law of Demeter [8], design & architecture patterns [2][3], and many other sources to provide us with criteria needed to construct a BBN. The qualitative part of SAABNet uses these general insights to arrange a set of approximately 30 variables in a graph. The graph captures such notions as "multiple inheritance affects complexity negatively" or "small components are good for flexibility". We have organized these variables into three categories:

- *Architecture Attributes*: these are basic properties of an architecture (e.g. depth of the inheritance tree, the programming language, etc.). Architecture Attributes are also suitable for incorporating metrics. However we did not use this in SAABNet.
- *Quality Criteria*: these are more abstract than Architecture Attributes and may include such things as complexity, coupling, etc.
- *Quality Factors*: These are on an even more abstract level. Quality factors are very general properties of a system (e.g. performance, maintainability and scalability). In SAABNet, quality factors can be found at the bottom of the graph. This means they are decomposed into less abstract quality criteria which in turn are decomposed into other criteria or into architecture attributes.

This categorization was inspired by McCall's quality requirement framework [4], though at several points we deviated from this model. In this model, abstract quality factors, representing quality requirements, are decomposed in less abstract quality criteria. We have added an additional decomposition layer (not found in McCall's model), called architecture attributes, that is even less abstract. The reason for this was that we needed a way to incorporate basic knowledge of the system into the assessment. Architecture attributes represent concrete, observable artifacts of an architecture and make it possible to include this type of knowledge.

A full specification of SAABNet is beyond the scope of this paper. Instead we refer to [6] for a more detailed description. In figure 2 an overview of the qualitative specification of SAABNet is presented. The quantitative specification of SAABNet adds some numbers to the general notions captured in the qualitative specification. While the qualitative specification can easily be associated with general knowledge about software architecture, this is not easily done with the quantitative specification. We have found that estimating probabilities for all the relations between the variables is a hard and error

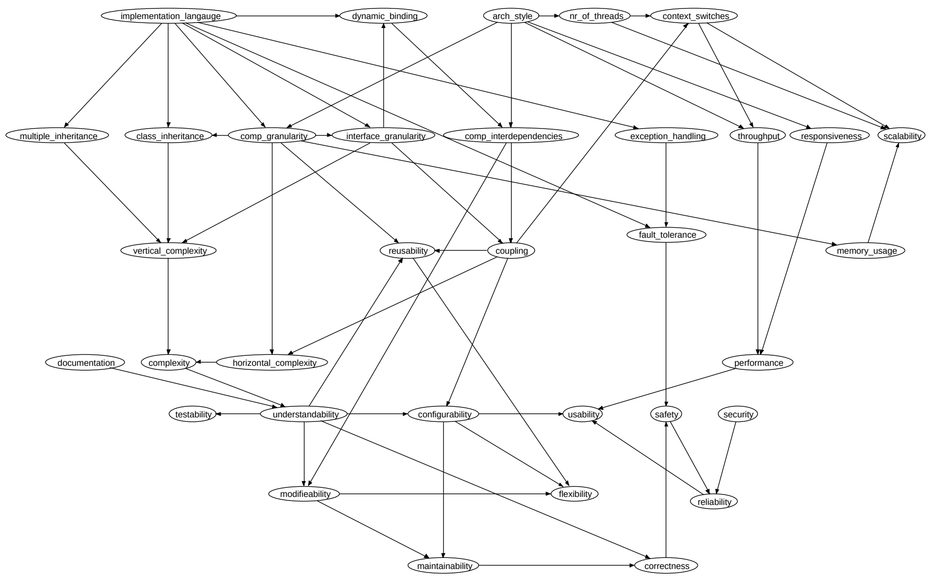

Figure 2 Qualitative specification of SAABNet

prone process. Existing research on BBN technology [11] suggests that this is a general problem in designing a BBN. In SAABNet we configured the network by running a number of cases and adjusting the initially guessed probabilities accordingly. We have to admit that at this point the network is not trained optimally. Such training would require input from more architecture experts and more test cases.

Usage

Currently SAABNet is implemented as a network specification for Hugin [9]. Hugin is a tool for designing and testing bayesian belief networks. It allows developers to draw the network with some simple tools. In addition it makes it easy to insert the quantitative data into the network. By running Hugin in the compiled mode, it is possible to interact with the network and test whether it works properly. In this mode, Hugin calculates a priori probabilities for each state in each node of the network based on the quantitative specification of the network. Evidence can be entered to the network by manually setting probabilities in the network. Each time evidence is entered, all the probabilities are recalculated. In BBN terminology this is called propagation of evidence through the network.

While Hugin makes it easy to read the output of the network by providing a graphical representation of the probabilities of each node as a bar graph, interpreting and explaining the output is the responsibility of the user. The general strategy of using a BBN is very simple: enter evidence for some variables, observe the effect of the evidence on other variables and try to explain the new probabilities. The structure of the network can be helpfull in this last step since the arrows in the graph express a causal relation. One has to keep in mind though that entered evidence propagates in both directions, even though the graph is directed.

In some cases the output of a BBN will conflict with what the user expects. Especially in those cases it is essential that the user understands why the BBN gives this output. Maybe the user had the wrong assumptions about the situation, maybe not enough evidence was entered (or too much), maybe the network is right or maybe there's a small error in the network (either in the qualitative or in the quantitative specification).

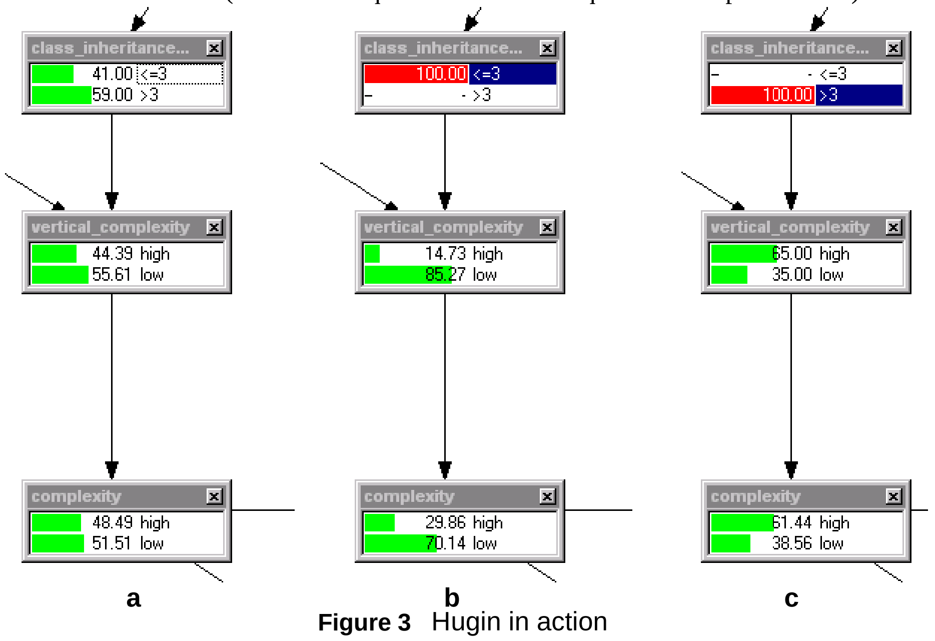

Figure 3 Hugin in action

In figure 3 a typical example of an interaction with Hugin is given. In the three screenshots, variable editing windows are projected on the variables in the SAABNet graph. We only display three nodes of the graph here but normally the rest of the graph is also visible. In figure 3a no information has been entered to the network. In figure 3b the depth of the inheritance tree is set to low (indicated by a red bar). This has some concequences in the two other displayed variables. In figure 3c the effect of setting the inheritance tree depth to a high value is shown. This simple example clearly shows how evidence propagates through the network along the arrows in the graph.

Though interacting with Hugin directly is a very powerful way of using the network, it is not really suitable for end users since Hugin is intended as a tool for BBN developers. End users cannot be expected to understand and appreciate all the details about a BBN. They need to be protected from its complexity. By using the Hugin API it is possible to write applications that interact with a Bayesian Belief Network. This API could be used to write SAABNet applications that help a user assess a software architecture. Such an application could for instance be integrated with design tools such as Rational Rose [10]. In our vision a standalone tool would be of little use to a designer since that would almost certainly require a developer to enter the same information in multiple places. An additional advantage would be that integration allows for automatic acquiring of information.

We have experimented with the Hugin API and found it easy to use. We created a simple GUI for SAABNet that allows one to enter data and read out the results. We also created a simple explain function built in the tool since we feel that one of the advantages of using a BBN is exploiting its structure to understand the output. Our efforts however did not result in a usable tool. We don't think a BBN should be used as a black-box since usually it is more important to understand why certain output occurs than the mere fact that it occurs. In the case of SAABNet, the intention is that software architects enter some data, read out the results and try to link those results to their own opinion. Explaining anomalies between the two helps understanding the architecture better.

Though SAABNet's output is quantitative (a priori probabilities), the main goal of SAABNet is not to be exact but to help understand how the variables interact. The general strategy is to enter information that is known or that needs verification and observe the recalculated variables. We identified four different strategies:

- *Diagnostic use*. One of the uses of SAABNet is that as a diagnostic tool. When using SAABNet in this way, the user is trying to find possible causes for problems in an architecture. Usually some architecture attributes are known and possibly also some quality criteria are known. In addition there are one or more Quality Factors which represent the actual problem. If, for instance, the implementation of an architecture has bad performance, the performance variable should be set to "bad".
- *Impact analysis*. Another way to use SAABNet is to evaluate the consequences of a future change in the architecture on the quality factors. To do so, the architecture attributes of the future architecture have to be entered as evidence. The network then calculates the quality criteria and the quality factors that are likely for such architecture attributes.
- *Quality attribute prediction*. In this type of use, as much information as possible is collected and put in the SAABNet. From this information, the SAABNet can calculate all the variables that have not been entered. This is ideal for discovering potential problem areas in the architecture early on but can also be used to get an impression of the quality attributes of a future architecture
- *Quality attribute fulfillment*. The first three approaches all required an architecture design. Early in the design process when the design is still incomplete, these approaches may not be an option. In this stage SAABNet can be used to help choose the architecture attributes. This can be done by entering information about the quality factors into SAABNet. The probabilities for all the architecture attributes are then calculated. This information can be used to make decisions during the design process. If, for instance, the architecture has to be highly maintainable, SAABNet will probably give a high probability on single inheritance since multiple inheritance affects maintenance negatively. Based on this probability, the design team may decide against the use of multiple inheritance or use it only when there's no other possibility.

These four strategies can be used simultaneously. The results of a diagnosed problem might for instance be used to do an impact analysis on a possible solution to these problems.

Conclusion

In this paper we presented SAABNet. We reflected on some problems with the traditional way of performing architecture assessment and argued that traditional methods fail early on in the development process due to a lack of measurable assets. We then presented a solution for this problem that does not rely on measurements but exploits qualitative knowledge. For future work we think that improving and extending SAABNet should have priority. Also building a user interface for end users may increase its usability. Furthermore we think that further validation (either in the form of an experiment involving end users or a larger case) is needed to prove our claims.

References

[1] J. Bosch, P. Molin, *"Software Architecture Design: Evaluation and Transformation"*, in Proceedings of the 1999 IEEE Conference on Engineering of Computer Based Systems. March 1999.

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

[3] E. Gamma, R. Helm, R. Johnson, J. Vlissides, *"Design Patterns - Elements of Reusable Object Oriented software"*, Addison-Wesley, 1995.

[4] J. A. McCall, *"Quality Factors"*, encyclopedia of Software Engineering, vol 2 O-Z pp. 958-969, John Wiley & Sons New York 1994.

[5] J. Pearl, *"Probabilistic Reasoning in Intelligent Systems"*, Morgan Kaufmann Publishers, Inc. San Mateo 1988.

[6] J. van Gurp, J. Bosch, *"SAABNet: Managing Qualitative Knowledge in Software Architecture Assessment"*, Proceedings of the 2000 IEEE Conference on Engineering of Computer Based Systems, p. 45-53, March 2000.

[7] S. R. Chidamber C. F Kemerer, *"A Metrics Suite for Object Oriented Design"*, IEEE Transactions on Software Enfineering, Vol. 20 no. 6, June 1994, pp476-493

[8] K. Lieberherr, I. Holland, A. Riel, *"Object-Oriented Programming: An Objective Sense of Style"*, in Proceedings of the 1988 OOPSLA Conference, San Diego, California, September 1988, pp. 323–334

[9] Hugin *"Hugin Expert A/S - Homepage"*, http://www.hugin.dk.

[10] Rational, *"Rational Software"*, http://www.rational.com/

[11] M. J. Drudzel, L. C. van der Gaag, *"Elicitation for Belief Networks: Combining Qualitative and Quantitative Information"*, Proceedings of the 11th Annual Conference on Uncertainty in Artificial Intelligence (UAI-95), pp. 141-148, Montreal August 1995.

[12] M. Neil, B. Littlewood, N. Fenton, *"Applying Bayesian Belief Networks to Systems Dependability Assessment"*, Proceedings of Safety Critical Systems Club Symposium, Leeds, Springer-Verlag February 1996.

[13] M. Neil, N. Fenton, "Predicting Software Quality using Bayesian Belief Networks", Proceedings of 21st Annual Software Engineering Workshop, 1996.