Fraunhofer Institut
Experimentelles
Software Engineering

Applying Bayesian Belief Networks for early software quality modeling

Authors:
Adam Trendowicz
Teade Punter

Supported by the
ITEA EMPRESS project

IESE-Report No. 117.03/E
Version 1.0
November 31, 2003

A publication by Fraunhofer IESE

Fraunhofer IESE is an institute of the Fraunhofer Gesellschaft.
The institute transfers innovative software development techniques, methods and tools into industrial practice, assists companies in building software competencies customized to their needs, and helps them to establish a competetive market position.

Fraunhofer IESE is directed by
Prof. Dr. Dieter Rombach
Sauerwiesen 6
D-67661 Kaiserslautern

Abstract

Bayesian Belief Networks (BBNs) are becoming popular within the Software Engineering research community. This report analyses the applicability of Bayesian Belief Nets to model quality of evolutionary software systems. The analysis is done by giving an overview of the domains where BBNs are currently applied. Then a process is introduces to apply BBNs in software quality modeling. After that a discussion on the applicability is conducted.

We conclude that Bayesian Belief Networks are profitable for modeling software quality of evolutionary software systems, because the complexity of such models can be made transparent with BBNs. However, BBNs could not be practical to apply for large models because of the great effort to build initial network.

This report is written as a contribution to Work package 3.5.2 of the ITEA EMPRESS project, in which Fraunhofer IESE was participating.

Keywords: Bayesian Belief Nets, quality modeling, evolutionary systems, non-functional requirements, software product lines

Table of Contents

1 Introduction

Embedded software already covers the majority of the software market and its production expands explosively. As a consequence, quality demands, especially in terms of non-functional requirements (e.g., dependability, maintainability), are of high importance and increase continuously. At the same time, software engineers are under more and more pressure to develop new systems in less time and at lower costs.

The approaches based on quality models seem to cope with producing high quality software at low costs. They allow starting quality evaluation at early stages of software development process and control it through the whole software lifecycle.

According to [ISO14598] the term **quality model** is defined as "the set of characteristics and relationships between them which provides the basis for specifying quality requirements and evaluating quality".

Despite the variety of existing quality models, they do not cover all the important aspects of quality modeling and evaluation. Many of them just replicate the lacks of others. For example the ISO 9126 quality model [ISO9126] defines a fixed set of quality characteristics. Nevertheless it is unrealistic to assume that it is possible to define a prescriptive view of necessary and sufficient quality characteristics to describe quality requirements for every project.

In [Trendowicz & Punter, 2003] and [Punter, Trendowicz and Kaiser, 2002] criteria were stated that are relevant when evaluating quality of software product lines, namely: flexibility, reusability and transparency. *Flexibility* is that the content and structure of the model are defined according individual needs and experiences of given organization so that resulting model is well custom-tailored to individual characteristics of the organization. *Reusability* is about reusing the model's content (quality characteristics, metrics) as well as structure (relationships and probabilities) for similar software products. *Transparency* of a quality model is needed to allow learning about quality relationships as well as easy identification of gaps, redundancies and conflicts to set relationships between the attributes.

The capacity of a quality model depends on the selection of the attributes (are the right attributes as well as the complete set of attributes selected to predict the quality) and the relationships set between: 1) the attributes and 2) attributes and metrics. Such modeling can result in quite complex relationships, where it is not always easy to get the overview, see for example Figure 1.

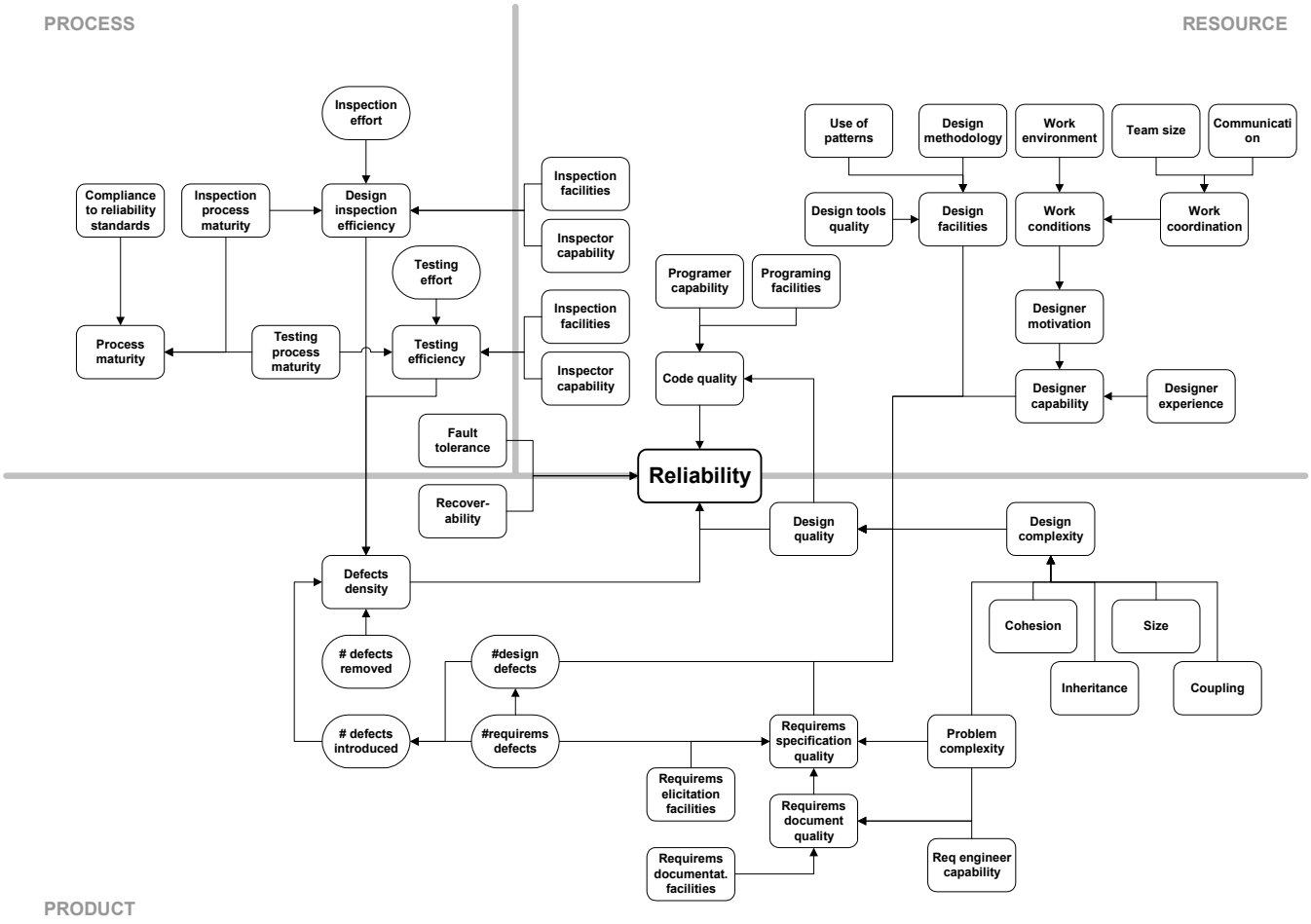

Figure 1 Example quality model for reliability

Another problem of quality models is the lack of sufficient information at the start of using the model. Often, only a subset of the relevant attributes is distinguished at the beginning of the modeling the quality. Starting with such an initial model might lead to the definition of other relevant attributes as well as to the refinement of the relationships. A third problem concerns how to involve different stakeholder views in a quality model. It is widely recognized that "quality is in the eyes of its beholders". However, less is known how to integrate these stakeholders in the quality model definition.

We think that each of these three problems (complexity, missing information and insufficient stakeholder involvement) will negatively influence the first two criteria of quality models (flexibility and reusability). To address this, we thought that an alternative way of modeling the relationships – by using Bayesian Belief Nets (BBNs) – between attributes and between attributes and metrics would be helpful. A quality approach using Bayesian Belief Nets to model quality relation-

ships seem to perfectly match to the nature of software engineering and current trends in software quality modeling. Encouraged with the properties of BBNs we decided to apply them to evaluate non-functional requirements in the context of evolving real-time embedded software systems. We designed the Prometheus[1] approach [Punter & Trendowicz and Kaiser, 2002] and deployed it in context of Empress project.

Several authors have advocated the use of BBN in Software Engineering. Fenton and others are in favor of using BBN when predicting defects as well as modeling dependability and reliability [Fenton & Nail, 1999], [Fenton et al., 2001]. They think that BBNs are able to represent and manipulate complex models that might never be implemented using conventional models. Therefore an easier understanding of chains of complex and seemingly contradictory reasoning via the transparent graphical format seems to be likely when modeling with BBN.

Besides that the Bayesian approach enables statistical inference to be augmented by expert judgment in those areas of a problem domain where empirical data (measurement) is sparse. BBNs do also explicitly model "ignorance" and uncertainty in estimates. The model can predict events based on partial (missing) or uncertain data. This enables decision support under uncertainty and might be relevant when dealing with conflicting stakeholder opinions.

The next paragraph explains the concept of BBNs. In paragraph 3 a simple example of BBNs' application is presented in order to explain the approach in more detail. In paragraph 4 we address experiences concerning the use of BBNs presented in available literature and combine them with our observations. Finally, in paragraph 5 we discuss the applicability of BBNs to model quality and we recommend the directions for further research.

[1] Probabilistic Method for Early Evaluation of Non-functional Requirements

2 Bayesian Belief Nets

A Bayesian Belief Net (BBN) or Bayesian net is a directed graph in which the nodes represent probabilities, while the arrows between the nodes represent dependencies. Each node in the graph may contain a number of states and has an associated set of Conditional Probability Distributions (CPD). Summarizing the BBN consists of:

- structure (topology) – the set of nodes and directed arcs. Nodes represent variables (events). The arcs represent the relationships between the variables (dependencies).
- content (parameters) – CPD related to each node (variable) of the graph.

Two special types of nodes are distinguished: *Root* is a node without parents and *leaf* is a node without children.

For discrete variables, the CPD can be represented as a Node Probability Table (NPT), which lists the probabilities that given node takes on each of its different values for each combination of values of its parents. For example, Figure 1 presents BBN in which all nodes are binary, i.e., each has only two possible values. For continuous variables, each CPD is represented as probability distribution e.g., most common Gaussian distribution. BBN might also contain nodes of different kinds. However, more advanced approaches have to be applied in that case [Olesen, 1993].

The construction of a BBN starts with the identification of the relevant variables in the domain that has to be modeled. After that the dependencies between the variables has to be determined. The third step is adding the conditional probabilities (CPDs) to the network. Then the initial network should be tested to verify if it reflects our intuitions regarding modeled quality.

After having constructed the BBN, it provides a mechanism for probabilistic inference. At the start, each node has its initial probability values (e.g., given by expert or excracted from experience database). When the new knowledge is available about probabilities (so-called *fact*), the appropriate NPT is altered and its impact is propagated over the whole graph (the whole network is recalculated basis on the new fact).

3 Domains of BBN-application

This section describes in which software engineering domains Bayesian probability approaches have been applied and which purpose they served.

BBNs have been applied on a large scale in the medical fields. They are used when dealing with diagnosis, treatment selection, planning and prognosis [Lucas, 2000][Lucas, 2001]. Bayesian networks are also widely used in software systems. The most well known application is probably in Microsoft applications, where BBNs underlie the help wizards in Microsoft Office or intelligent" printer fault diagnostic system. Nokia Networks uses the BBNs in a prototype tool for efficient diagnosis of mobile networks. By having an automated tool that reads network performance data and from that estimates and monitors network problems ranked by probability, the network operator gets an efficient troubleshooting procedure saving both expensive expert resources and downtime of the network [Barco et al., 2002]. Finally BBNs became the subject of interest of software quality evaluation community.

Littlewood [Littlewood et al., 2000] applies BBNs to evaluate software safety. BBNs support the "What-if" analysis i.e. entering various possible combinations of observations in order to see their influence on software safety. For example, "What-if" analysis can answer the questions like: "What would I need to observe, at each successive stage of project, in order to show that the chances of success are still acceptable – that the project is in track?"

Delict et al [Delic et al. 1997] as well as Fenton [SERENE, 1999] [Fenton, 1996], [Fenton et al., 2001 use Bayesian networks to evaluate software dependability. Fenton for example combines into Bayesian model different product, process and resource factors (e.g., testing quality, team competence) in order to evaluate the number of defects found during testing.

[Fenton and Neal, 2001] come up with the following characteristics that plead for the use of BBN in the SE-domain. BBNs can easily model causal influences between variables in a specified domain. This specification of complex relationships using conditional probability statements is done in a graphical way so that easier understanding of chains of complex and seemingly contradictory reasoning will be possible. The Bayesian approach does also enable statistical inference to be augmented by expert judgment in those areas of a problem domain where empirical data (measurement) is sparse. A third advantage is that BBNs can deal with events based on partial (missing) or uncertain data, so that decision support under uncertainty is possible. Rodriguez et al [Rodriguez et al, 2003] add that Bayesian systems model probabilities rather than exact value,

which means that uncertainty, can be handled effectively and represented explicitly. Besides there are software tools that facilitate modeling with Bayesian networks. These arguments make [Fenton and Neal, 2001] to conclude that BBNs are able to represent and manipulate complex models that might never be implemented using conventional modeling approaches.

Gurp and Bosch have developed a BBN to evaluate software architectures [Gurp & Bosch, 1999. The SAABNet (Software Architecture Assessment Belief Network) supports a qualitative assessment during architecture design phase of software development. They apply the quality factors (e.g., reliability, maintainability) and criteria (complexity, fault tolerance) from McCall's Factor-Criteria-Metric model. However instead of metrics the last layer in the model defines architecture attributes (class coupling, class inheritance). The network has been applied to automate the reasoning of qualitative knowledge in a software development process. The BBN serves as an assisting technique to the designer expertise. Despite of the small size of the belief network, the authors argue that they were able to get meaningful output from it in the cases were it was tested (a PDA system, and an architecture for an embedded system) [Gurp and Bosch, 1999]. However, from the paper it is not clear if the outcomes have been tested for their correctness. Gurp and Bosch think that the SAABnet will assist in selecting system properties based on quality requirements. When a flexible, highly configurable system is needed without taking a performance penalty, the network will suggest what states the other variables need to be in to make this possible. This does not automatically mean that other values of these variables will not give the same result, it just means that it is not as likely to happen. Another possibility to use the network is when verifying if certain properties for the variables indeed has the wanted effect on other variables. This is useful to provide argumentation for decisions early in the design process. A third possibility to apply SAABnet is for identifying variables that will need special attention during the development process.

Rodriguez et al [Rodriguez et al, 2003] focus on the different type of algorithms to calculate Bayesian Networks. They propose four classifiers (algorithms) to calculate the nets, namely: General Bayesian Network, Naïve Bayes, Tree Augmented Naïve and Forest Augmented Naïve Bayes classifier. The authors found that the FAN classifier should produce better results since it can generate a broader range of network structures and in turn, better estimate actual probability distribution. However, their data set size was too small and therefore it could not effectively illustrate the merits of one over another.

BBNs have been applied to software engineering, by several authors with several purposes. The work of Fenton et al provides significant contributions to the software engineering domain. The work of Gurp and Bosch is closely related to our purpose to apply BBN for modeling software quality in early and evolutionary systems.

4 How to apply BBNs for quality modeling?

To apply Bayesian Belief Nets for Quality modeling we need to distinguish between the learning from the model and the act of defining an appropriate BBN. The learning about the model is addressed by the elements of a model (section 0) and propagating through the model (section 4.2). Section 4.3 presents an approach how to define a BBN for quality modeling purposes.

4.1 Elements of the model

This section provides the elements of a software quality model when modeled as a BBN.
The part of quality model that is presented in Figure 2 is an example of a BBN for "reliability prediction", which includes also the NPTs.

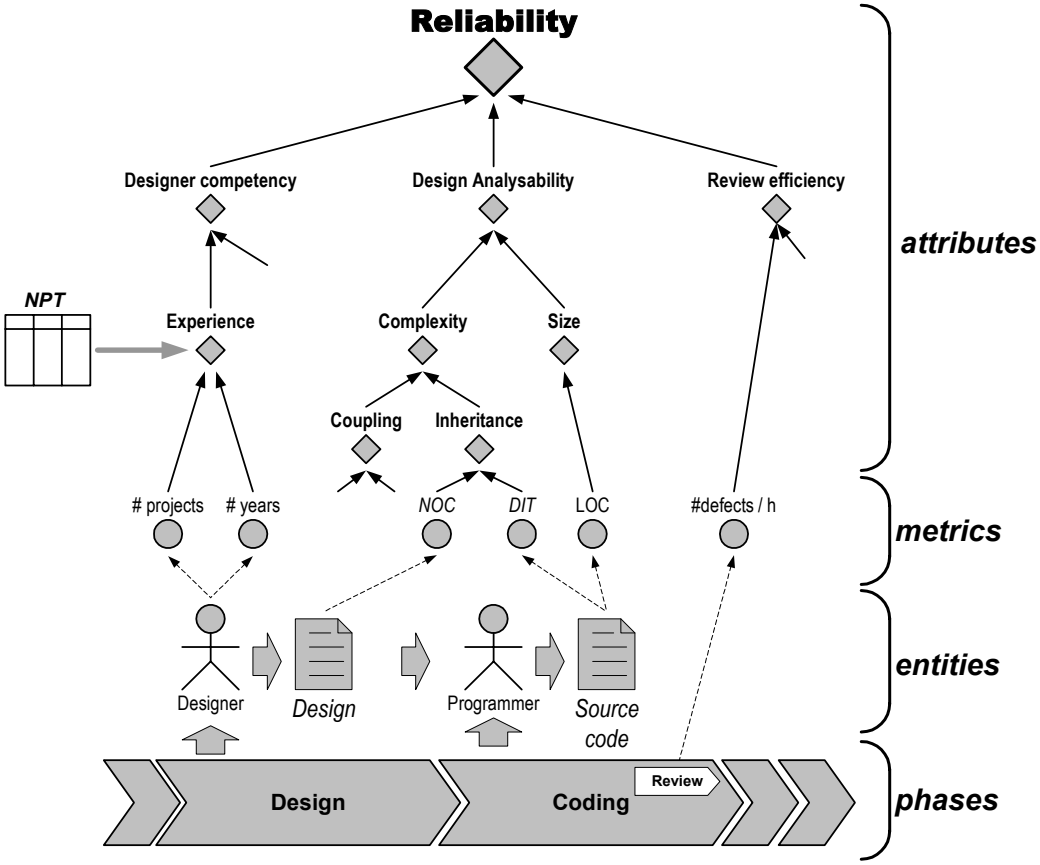

Figure 2 Example BBN for reliability.

The nodes represent attributes (discrete or continuous), for example, the node "Experience" is discrete and can have three values "low", "medium" and "high" whereas the node "size" might be continuous (such as LOC). Dependent of the organizations capabilities and needs it is of course possible to represent as discrete the variables which are in principle continuous. For example we might represent size of code as "small", "medium" and "large". The arcs represent influential relationships between attributes. The number of similar projects he/she participated in and years of experience define designer experience. The exemplary node probability table (NPT) for "Experience" node might look the one shown in Table 1. For the simplicity of the example we have made all considered in NPT nodes discrete so that each of them take on just three discrete values. The NPTs capture the conditional probabilities of a "Experience" node based on given the state of its parent nodes.
As we already mentioned there may be several ways of determining the probabilities for the NPTs. One of the benefits of BBNs stems from the fact that we are able to accommodate both subjective probabilities (elicited from domain experts) and probabilities based on objective data.

# similar projects		few			many			lot		
years of experience		few	many	lot	few	many	lot	few	many	lot
experience	Low	0,70	0,50	0,33	0,50	0,33	0,20	0,20	0,33	0,70
	Med	0,20	0,30	0,33	0,30	0,33	0,30	0,30	0,33	0,20
	High	0,10	0,20	0,33	0,20	0,33	0,50	0,50	0,33	0,10

Table 1 Example Node Probability Table

Having entered the initial probabilities we can calculate initial estimation of interesting us quality (here reliability) and update actual probabilities (propagation) through the whole development process when the new knowledge appears (facts). For example after implementation phase we can measure the actual size of the source code. Therefore the attribute "size" is then a fact what is reflected by putting into NPT measured value (let's say *Size=small*) with probability 1 and other possible values (*Size=medium*, *Size=large*) with probability 0. After recalculating NPTs in all remaining nodes we obtain new, more accurate, evaluation of reliability. More detailed explanation of propagation mechanism is presented in the following sub section.

4.2 Probabilistic inference

The probabilistic inference is actually the propagation though the Bayesian net and represents the learning about the beliefs in the net. Probabilistic inference is illustrated by the following example (Figure 3).

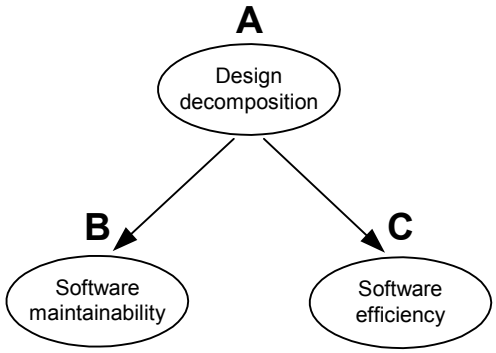

Figure 3 Example of Bayesian Belief Net

The dependency graph represents the common belief of software engineers that decomposition of software design (node A) is related (influences) the maintainability (node B) and efficiency (node C) of the final software product. In order to complete the BBN we have to extend our structure by adding probabilities (beliefs). Before that we need to quantify all variables. For the simplicity of the presentation let's assume that all variables have a discrete character and that each of them has only two values, which are denoted by *Small/Large* for the node A and *Low/High* for nodes B, C. Now we can put our initial believes into probability tables (Table 2-4).

Design decomposition	**Probability**
Small	0.7
Large	0.3

Table 2 Node probability table for *Design decomposition* variable.

		Design decomposition	
		Small	Large
Software maintainability	Low	0.7	0.2
	High	0.3	0.8

Table 3 Node probability table for *Software maintainability* variable.

		Design decomposition	
		Small	Large
Software efficiency	Low	0.3	0.9
	High	0.7	0.1

Table 4 Node probability table for *Software efficiency* variable.

The tables 2 to 4 contain so-called initial probabilities and reflect the beliefs about the chance of high or low software maintainability and efficiency. We already feel (by intuition) that design (and in consequence whole software) decomposition has positive influence on software maintainability and negative influence on software efficiency (Tables 3, 4). Therefore, for example, we believe there is a low probability that software would be of high efficiency given it is of large decomposition. Let's assume that we are developing an embedded real time system. In that context efficiency is more important than maintainability so assuming that developers know about mentioned relationships and we can belief that software design would be of small decomposition (Table 2).

Having entered the probabilities we can now use Bayesian probability to conduct various types of analysis. For example, we might want to calculate the (unconditional) probability that software would be of high maintainability:

*P(High maintain.) = P(High maintain. | Large dec.) * P(Large dec.) +*
*+ (High maintain. | Small dec.) * P(Small dec.)*

*p(High maintain.) = (0.8 * 0.3) + (0.3 * 0.7) = 0.45*

This is called the marginal probability. Similarly, the marginal probability of high software efficiency would be 0.52.

The most important use of BBNs is in revising probabilities in the light of actual observations of events. Suppose, for example, that we know the design is of *small decomposition*. In this case we can enter the evidence that *Design decomposition = Small*. The conditional probability tables already tell us the revised probabilities for *High software maintainability* (0.3) and *High Software efficiency* (0.7).

Suppose, that we do not know if the design decomposition is large or small. Nevertheless, after evaluating some components, we do know that its efficiency is high. Then we can enter the evidence that *Software efficiency = High* and we can use this observation to determine:

1. the (revised) probability that the design decomposition is large; and

2. the (revised) probability that software maintainability is high.

To calculate 1) we use Bayes theorem:

$$p(Large\ dec. \mid High\ eff.) = \frac{p(High\ eff. \mid Large\ dec.) \times p(Large\ dec.)}{p(High\ eff.)}$$

$$p(Large\ dec. \mid High\ eff.) = \frac{0.1 \times 0.3}{0.52} = 0.0577$$

Thus, the observation that efficiency is high significantly decreases the probability that software is of large decomposition (down from 0.3 to 0.0577). Moreover, we can use this revised probability to calculate probability of high software maintainability:

*P(High maintain.) = P(High maintain. | Large dec.) * P(Large dec.) +*
*+ P(High maintain. | Small dec.) * P(Small dec.)*

*p(High maintain.) = (0.8 * 0.0577) + (0.3 * 0.9423) = 0.3288*

Thus, the observation that efficiency is high decreased the probability that it would be of high maintainability. When we enter evidence and use it to update the probabilities in this way we call it *propagation*.

4.3 Defining a BBN-based quality model

This section deals about how to start in modeling a BBN for a quality model.

The initial quality model is basically set up by defining attributes, possible relationships and adding probabilities to these relationships. The following steps have to be taken during a group session (steps 1 to 3) and a setting an initial BBN (step 4).

1. Define a quality goal (or quality goals) for the software – this is done according to the goal measurement template, which defines: object, purpose, quality focus, viewpoint and context.
2. Specify quality characteristics – this step concerns the refinement of the quality goals into a set of quality characteristics and sub (or sub-sub) characteristics. This procedure goes on as long as there is a set of measurable characteristics defined. A sub-characteristic is measurable when it is possible to attach it to a particular component and define one or more corresponding metrics. References on factors that potentially influence the quality goal might be useful to support this specification step. The result of this step is a set of attributes and their associated measures.
3. Specify relationships – relationships among the quality characteristics and between the quality attributes and their measures are set. Apart of the existence of a relationship between attributes and/or attributes and measures, also the probability that such a relationship will appear is asked. These probabilities are set as the importance of the defined qual-

ity characteristics (weights), being a relative importance of the quality factors.

4. Set initial BBN – this activity transfers the results of steps 1 to 3 into an initial BBN that expressed the quality model with its probabilities for the software. The probabilities are set by a moderator by using the table presented above.

During the group session people involved in the software development but having different roles and responsibilities in the development should be involved. We think that at least people with the following roles should be represented for specification in an embedded software context: architect, engineer and product manager. A moderator leads the group session and he/she is responsible for the defining the initial BBN too. The results (of steps 2 and 3) are expressed in a table, like the following:

Quality characteristic	Related Quality characteristic	Weight of relationship (3 point scale)	Related Measures	Weight of characteristic – measure relationship
Q1	Q3	1	M1.1	2
Q2	Q4		M2.1, M2.2	M2.1: 3, M2.2: 1
Q3	Q1	1	M3.1, M3.2	M3.1:1, M3.2: 2
Q4				

Table 5 Example table with initial quality characteristics for BBN

In order to combine within the model the view of more software project stakeholders such as developer and maintainer additional mechanisms have to be applied.

First, all interested parties have to agree with regard to the set of quality attributes within the model. The brainstorming session could deal with this problem.

Second, NPT for each network node has to be filled. In principle nodes and NPTs are in the one-to-one relation what means that only one NPT is defined at each node. Therefore one possible solution to define NPTs could be joint session of all interested parties during which common probabilities are discussed and put into NPTs. However, this approach should be restricted to small networks where number of NPTs and required initial probabilities to define is relatively small. In case of larger nets, all interested parties could be introduced into the process of filling NPTs during joint session and fill the table individually afterwards. The results could be then combined by simply calculating average for each probability.

5 Discussing the applicability of BBNs

In this section the application of BBNs for quality modeling is discussed.

Our observations on the applicability of BBNs restrict to a discussion, because we could not find a demonstrator at one of the EMPRESS participating companies to apply BBN-based quality modeling. The discussion is conducted by addressing the three problems for quality models that were addressed in section 1, namely: complexity, missing information and stakeholder involvement.

Complexity

The identification of the model content and structure confirmed the common opinion that BBNs provide transparent and intuitive structure and content (in contradiction to statistical or artificial intelligence methods). The understandable composition of BBN allows not only analyzing and directly interfering the model but also learning about complex quality dependencies. The learning process is supported by BNNs through the whole software lifecycle. During the development of the initial network we can learn from the views of other stakeholders. Then, it is possible to modify the network if it does not reflect our intuition or experiences. In successive versions of a belief network, arcs could be added or discarded; nodes that prove to be useless could be eliminated. Other variables can be added when it is believed that they will have predictive power, or allow the validation of the argument through observation. The iterative process of building and analyzing BBNs is useful for clarifying otherwise informal reasoning, discarding weak arguments and warning of counterintuitive implications of seemingly obvious arguments. Organization can reuse the belief networks produced for a first application as templates for use in later (entirely or partially), in similar projects, also reducing at the same rime the cost of this approach.

Dealing with missing information

Applying probabilities and Bayesian theory in the field of quality modeling could be perceived both as a strength and weak point of the approach. The weakness is that output of such a model is probability that given quality will reach a certain value instead of the value itself. It means that BBN quality model could be in fact used to at most assessing quality risks instead of predicting the value of interesting us characteristics. However, it seems to be perfectly enough for the purpose of decision support. Anyway, software engineers often operate on dichotomized variables. For example efficiency expressed by response time is a continuous variable. From the perspective of fulfilling the non-functional requirements we are often not interested in an exact value of the response time, but more in an indication, if it is the response time is short enough. Probably we

would add some intermediate values like "acceptable" but that's enough for us to evaluate response time.

Stakeholder involvement

Operating on probabilities brings several advantages. First, it is possible to employ qualitative experts' knowledge to set up initial probabilities. Expert knowledge is a significant part of quality models at early stages of software life cycle where hardly few measurable artifacts are available. In organizations of a low maturity, where measurement does not exist and project success depends on human skills and experts' knowledge is the only basis for quality modeling.

A significant drawback of BBNs is that initial net requires from experts a great effort to put initial beliefs into node probability tables. For each discrete variable N and A_1, A_2…, A_i variables that influence N (parents of N), the number of initial conditional probabilities to be defined is equal to:

$X = n*a_1*a_2*…*a_i$

where n, a_1, a_2,…, a_i is the quantity of possible values for variable N, A_1, A_2…, A_i

For example, let's assume that each attribute within the network presented in Figure 4 has three possible values (small, medium, large).

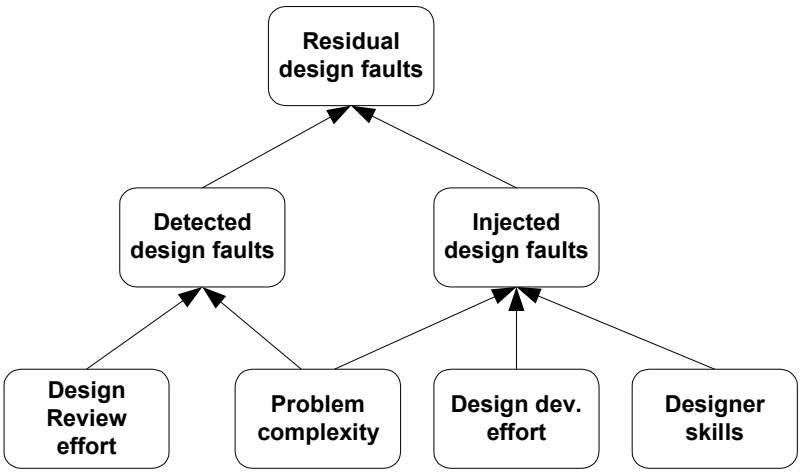

Figure 4 Example BBN for evaluating residual design faults

The NPTs that have to be filled include in total 7 tables:

- Four tables for root nodes (3 cells each)
- One table for "detected design faults" node (27 cells)
- One table for "injected design faults" node (81 cells)
- One for leaf node residual design faults" node (27).

Therefore, 147 cells should be filled with initial probabilities. In addition we have to be aware that the time required to introduce all the probabilities is usually not proportional to the number of the cells but also depends on the size of the tables. From the perspective of cognitive complexity it is probably easier for expert to embrace mentally several smaller tables (one by one) than one large table with many dimensions.

As shown before, the number of cells for probability tables in a given BNN grows exponential with the growth of the overall number of variables and the number of relationships. This fact imposes serious limitations on the size of BBN. However, the great effort required from experts to input initial probabilities could be reused in other software projects. Therefore the average effort on one BBN application decreases with the growing number of applications. Additionally the more similar is the project to some of the previous one the greater part of the BBN can be reused and the less effort is required from expert's side.

Both limiting the size of network and structuring it in a way that a maximal number of parents does not exceed 2 could also minimize the effort expenditure required to feed the model with conditional probabilities. For example the number of characteristics can be limited to the most influential ones and some intermediate (possibly) "artificial" nodes could be created in order to limit the number of node parents (see Figure 5 and Figure 6)

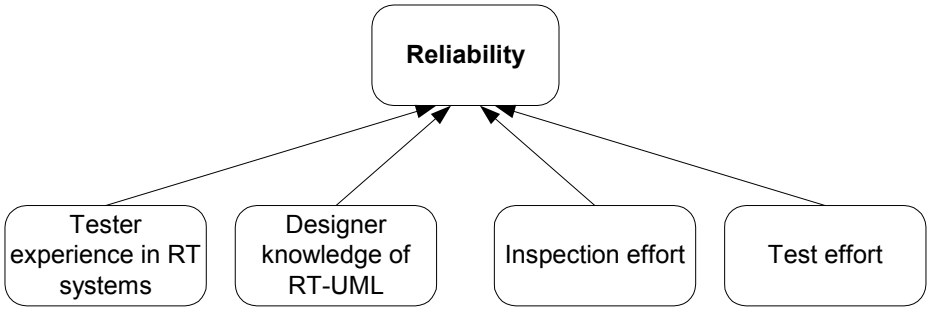

Figure 5 Example of node with many parents

Exponential growth of the number of probabilities entails also the exponential growth of the computational power required to (re-) calculate the network. Actually computing Bayesian probabilities in BBNs is a NP-Hard problem (which cannot be solved in polynomial time) [Cooper, 1990]. However, recently invented efficient algorithms let us neglect this problem if the net is not really huge.

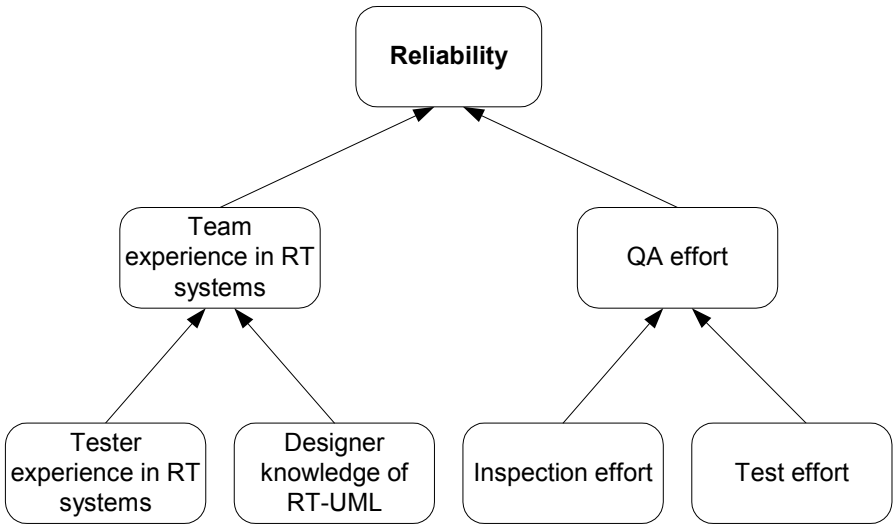

Figure 6 Network from Figure 5 after limiting the number of parents

Theoretically, BBN allows mixing discrete and continuous variables within one, hybrid BBN. In practice there are some limitations. Just recently Lerner and his colleagues presented a first inference algorithm for hybrid BBNs that allows discrete children of continuous parents [Lerner at al., 2001]. However it does not have practical repercussions because existing BBN tools does not yet support it. They either support hybrid nets excluding discrete children of continuous parents or does not support continuous variables at all. In practice, even if continuous variables are supported by a tool most of tools provides only one – Gaussian – distribution.

There is another problem with the continuous variables in the area of software engineering. Standard probability distributions known from social sciences often does not apply to software engineering phenomena. Additionally, most of software organizations do not have enough data from past projects to establish such distributions. However, as we have already mentioned software engineers are used to think in the discreet manner even about variables that are continuous in nature.

During our application we were not able to apply the model to assess its accuracy. Nevertheless, other researchers [Fenton et al., 2001] report that a significant amount of data is needed in order to make reasonably precise predictions for a specific project. In their study Fenton et al. observed that a disadvantage of a reliability model of higher complexity is the amount of data that is needed to support a statistically significant validation study.

6 Conclusions and further investigation

Recent research on the applicability of Bayesian Belief Nets is promising for solving the problems that existing quality modeling approaches face. BBNs have many characteristics that made them applicable for quality modeling. The dependency graphs and the feature to change the probabilities make them well tailorable for individual software project characteristics as well as reusable for other projects. The graph-based structure supports the transparency that is required by quality models. Modeling quality attributes with a graph facilitates the learning about interdependencies amongst quality characteristics. On the other hand, application of Bayesian probability let us put qualitative quality estimations in sound mathematical frames.

However we found also drawbacks of applying BBNs for quality modeling. The main drawback is the initial effort that is required to build a first version of a quality model. In section 5 it was discussed that filling the node probability tables require a great effort from experts. However the effort could be minimized when the network has a sound size and structure. Moreover, when the network is defined, it can be reused in subsequent projects and thus limit substantially the required costs of quality modeling.

A minor problem is the lack of tools to support BBNs with mixed discrete and continuous variables, and NP-hard computational complexity of BNN. It was mentioned in section 5 that software engineers operate mainly with discrete variables therefore exact evaluations with continuous variables are not necessary. On the other hand capabilities of computers and new effective algorithms to operate Bayesian networks let us neglect the computational complexity problem event for quite large networks.

Further research on BBNs should focus on how to support experts/stakeholders with efficient definition of initial NPTs.

References

[Barco et al., 2002] R. Barco, R. Guerrero, G.Hylander. L. Nielsen, M. Partanen, S. Patel, " Automated Troubleshooting of Mobile Networks using Bayesian Networks", proceedings of the IASTED International Conference on Communications Systems and Networks, September 9-12, 2002, Malaga, Spain (Calgary, Canada: ACTA Press, 2002), pp 105—110

[Cooper, 1990] GF.Cooper, "The computational complexity of probabilistic inference using Bayesian belief networks" (research note), Journal of Artificial Intelligence. vol. 42 pp 393-405, 1990

[Delic et al., 1997] K.A. Delic, F. Mazzanti, L. Strigini, "Formalising engineering judgement on software dependability view belief networks", DCCA-6, Sixth IFIP International Working Conference on Dependable Computing for Critical Applications, "Can we rely on computers", Garmish-Partenkirchen, Germany, 1997

[Fenton & Pfleeger, 1996] N.E. Fenton S.L.O.Pfleeger, „Software Metrics: A Rigorous and Practical Approach",PWS, 1998 (originally published by International Thomson Computer Press, 1996), ISBN 0534-95429-1

[Fenton & Neil, 1999] N.E.Fenton, M.Neil, „A Critique of Software Defect Prediction Models",IEEE Transactions on Software Engineering, vol. 25, no. 5, pp. 675-689, 1999

[Fenton et al., 2001] N.E.Fenton, P.Krause, M.Neil, *A Probabilistic Model for Software Defect Prediction*, accepted for publication IEEE Transactions in Software Engineering, 2001

[Fenton & Neil, 2001] N.E. Fenton, M. Neil, *Making Decisions: Using Bayesian Nets and MCDA*, Knowledge-Based Systems 14, pp. 307-325, 2001. Software Metrics: Uncertainty and Causal Modeling. Fenton N, Neil M and Krause P. EuroSPI conference, Limerick Institute of Technology, Limerick, 10th-12th October 2001.

[Gurp & Bosch, 1999] J. Gurp, J. Bosch, "Using Bayesian Belief Networks in Assessing Software Designs", ICT Architectures '99 , Amsterdam November 1999

[ISO9126, 2001] *Software Engineering - Product quality. Part 1: Quality model*, ISO/IEC International Standard, 2001

[ISO14598, 1999] ISO/IEC 14598 International Standard, Standard for Information technology - Software product evaluation - Part 1: General overview, 1999

[Khoshgoftaar & Allen, 1998] T.M. Khoshgoftaar, E,B. Allen, "Neural networks for software quality prediction", In W. Pedrycz, J.F. Peters, editors, Computational Intelligence in Software Engineering, 16:33-63 of Advances in Fuzzy Systems - Applications and Theory. World Scientific, 1998,

[Khoshgoftaar & Allen, 1999] T.M. Khoshgoftaar, E.B. Allen, "Logistic regression modeling of software quality", International Journal of Reliability, Quality and Safety Engineering 6 (4):303-317, 1999

[Lerner et al., 2001] U. Lerner, E. Segal, D. Koller, *Exact Inference in Networks with Discrete Children of Continuous Parents*. Uncertainty in Artificial Intelligence: Proceedings of the Seventeenth Conference (UAI-2001), pp. 319-328, Morgan Kaufmann Publishers, San Francisco, CA, 2001

[Littlewood et al., 2000] B. Littlewood, L. Strigini, D. Wright, N. Fenton, M. Neil, "Bayesian Belief Networks for Safety Assessment of Computer-based Systems", in System Performance Evaluation Methodologies and Applications (Ed: Gelenbe E), ISBN 0-8493-2357-6, pp. 349-364, CRC Press, Boca Raton, 2000

[Lucas, 2000] P. Lucas, "Bayesian model-based diagnosis", Technical Report, 2000; published in the International Journal of Approximate Reasoning 2001; 27(2): 99-119

[Lucas, 2001] P. Lucas, "Bayesian networks in medicine: a model-based approach to medical decision making", K-P. Adlassnig (ed.), Proceedings of the EUNITE workshop on Intelligent Systems in patient Care, Vienna, Oct. 2001, pp. 73-97

[Olesen, 1993] K.G. Olesen, "Causal Probabilistic Networks with Both Discrete and Continuous Variables", IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 15, no. 3, 1993, pp. 275-279

[Prowell et al., 1998] S.J. Prowell, C.J. Trammell, R.C. Linger, J.H. Poore, *Cleanroom Software Engineering. Technology and Process*, Reading: Addison-Wesley, 1998, xv, 390 S. ISBN: 0-201-85480-5

[Punter et al., 2002] T. Punter, A. Trendowicz and P. Kaiser „Evaluating Evolutionary Software Systems“, 4th International Conference on Product Focused Software Process Improvement, PROFES 2002, Rovaniemi (Finland), December 9 - 11, 2002.

[Rodriguez et al, 2003] D. Rodriguez, J. Dolado, M. Satpathy, Bayesian Networks and Classifiers in Project Management, in: Proceedings of 2nd workshop on Empirical Studies in Software Engineering (WSESE 2003), 2003.

[SERENE, 1999] SERENE consortium, SERENE (SafEty and Risk Evaluation using bayesian Nets): Method Manual, ESPRIT Project 22187, http://www.csr.city.ac.uk/people/norman.fenton/serene.htm, 1999.

[Trendowicz & Punter, 2003] A. Trendowicz, T. Punter, “Quality modeling for Software Product Lines”, 7th ECOOP workshop on Quantitative Approaches in Object-Oriented Software Engineering QAOOSE 2003, Darmstadt, Germany, July 2003

Document Information

Title:	Applying Bayesian Belief Networks for early software quality modeling
Date:	November 31, 2003
Report:	IESE-117.03/E
Status:	Final
Distribution:	Public