Proceedings of the International Multiconference on
Computer Science and Information Technology, pp. 351 – 360

ISSN 1896-7094

Numerical Assessment of Software Development Tools in Real-Time Safety Critical Systems Using Bayesian Belief Networks

Janusz Zalewski[1] , Andrew J. Kornecki [2], Henry L. Pfister [3]

[1] Florida Gulf Coast University
Fort Myers, FL 33965-6565, USA
zalewski@fgcu.edu_
[2] Embry Riddle Aeronautical University
Daytona Beach, FL 32114, USA
kornecka@erau.edu_
[3] Air Force Research Laboratory
Eglin AFB, FL 32542, USA
pfister@eglin.af.mil_

Abstract. This paper explores the applicability of Bayesian belief networks (BBN's) to assess software development tools for real-time safety-critical applications. Bayesian inference rules are applied to experimental results from a previous study, including such properties of safety-critical software as efficiency, usability, functionality and traceability, and a numerical assessment of a tool is derived to assist in the tool qualification process.

1 Introduction

Software development tools for real-time safety-critical systems are recently receiving increased attention of researchers, due to their significance in developing embedded applications. In certain industries, such as automotive, avionics, and military, heavily relying on ensuring safe operation of software controllers embedded in their products (trains, cars, aircraft, unmanned aerial vehicles, etc.), it becomes more and more common to take a closer look at the tools used to develop software for these controllers. Certification authorities are also investigating the necessity of potential tool qualification and problems related to this process.

In a recent study [9,10], a software tool taxonomy has been developed and the key criteria for assessment of software development tools used in real-time safety-critical applications have been identified. The criteria, when associated with respective metrics, allowed semi-objective comparative analysis of tools for which performance data have been collected. However, the overall assessment of the tool, say, for qualification purposes, requires a more precise, numerical estimate of the tool quality according to these predefined criteria. In this view, a more comprehensive method of tool assessment is needed.

Assessing software quality, in general, and particularly in safety-related applications, has always been a problem, mostly because of a scarcity of long-term statistical data on which the assessment can be based. Using more technical terms, the problem is related to insufficient information and uncertainty associated with the available data. However, in the last decade, there have been a number of publications related to the assessment of various quality attributes of software for critical applications, relying on non-statistical data. One interesting technique of that sort, which does not require a wealth of statistical information about the subject, is called Bayesian belief networks (BBN's). Examples of respective works include using BBN's to assess general software quality [11], software dependability [4], software safety [3], and software reliability [6].

The objective of the present work is to study the application of BBN's to the assessment of software development tools, based on the experimental data collected during the course of a previous project [9,10]. The rest of the paper is organized as follows. In Section 2, the principles of Bayesian belief networks and their

application to software quality assessment are briefly presented. Section 3 outlines the general problem of tool assessment in real-time safety-critical systems, and presents the application of BBN's to numerical evaluation of such tools. The preliminary results are outlined in Section 4.

2 Bayesian Belief Networks

2.1 Basic Information

A Bayesian belief network essentially relies on applying a Bayesian inference to a graphical representation of a problem in a form of a hierarchical network (Fig. 1). The annotated nodes represent random variables and directed arcs represent probabilities (beliefs) about the dependencies (relationships) among these variables. The numbers in nodes represent percentages of beliefs that a variable is in a particular state. Formally speaking, a network takes the form of a *directed acyclic graph* [8]. Each node has a number of states, with some probability distribution over the states. Each arc represents a conditional probability reflecting dependencies on the predecessor nodes. In the simplest case, the states could be just *true* and *false*, in a more complex case, the set of states may involve several discrete states (for example, low, medium, high, very high), and in the most general case, the domain of states can be continuous.

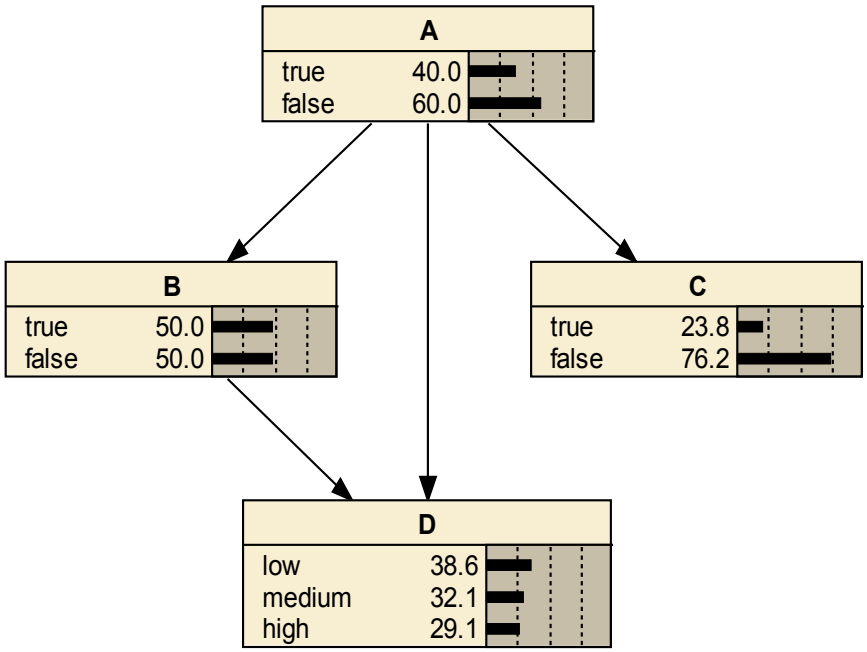

Fig. 1. Basic illustration of a BBN

A BBN is intended to show the causal relationships and thus allows the analyst to conduct inference about changes in probability distributions of certain variables (parameters), once values of some observable variables become known. In other words, based on some knowledge of events (i.e. available evidence) acquired at the leaf nodes (which may change their probability distribution), applying Bayesian reasoning one can deduct what might have caused such events to occur. This is called backward reasoning, and a specific example for the BBN from Figure 1 and some evidence obtained for node D, is shown in Figure 2. One can also do forward reasoning, in case respective evidence becomes available for the root nodes.

This kind of reasoning is based on Bayes' Theorem and is normally cumbersome and time consuming, if done by hand, but can be easily automated with software tools that became available throughout the 1990's.

There is a multitude of BBN tools to choose from, but the most commonly applied in software quality related problems are: AgenaRisk [1], BUGS [2], Hugin [7], and Netica [12] (with which the diagrams presented in this paper have been produced).

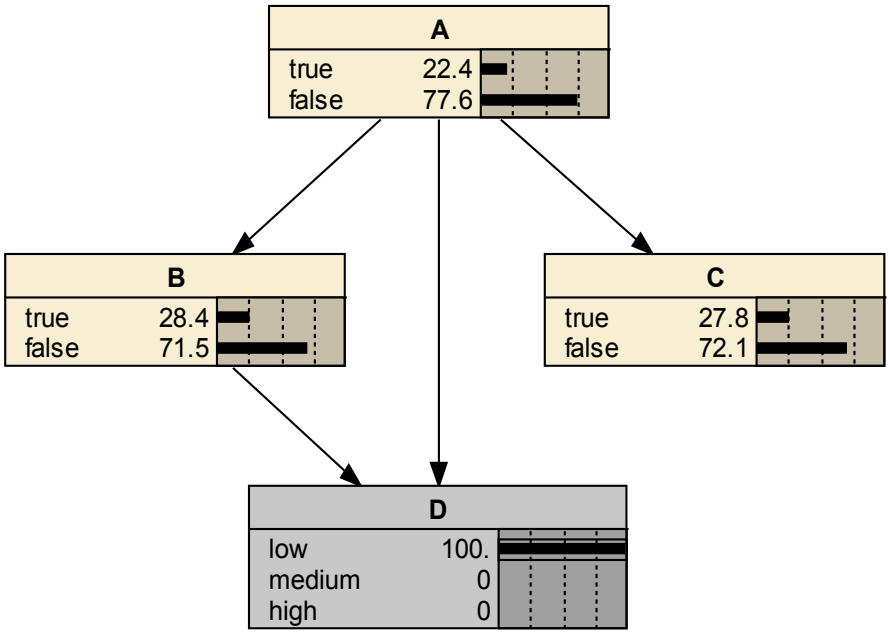

Fig. 2. Demonstration of an impact of evidence on the predecessor variables

2.2 Application of BBN's to Software Assessment

In building the BBN model, there are always two questions that have to be addressed: (1) how to organize the network of nodes and links, and (2) what probabilities assign to them. Below, we will discuss briefly how this is done in some of the studies related to software assessment published over the past decade.

Software Quality in General. In one of the first studies reported, Neil and Fenton [11] addressed the eternal question: "Can we predict the quality of our software *before* we can use it?" by applying BBN's to evaluate the *defect density* as a measure of software quality. A simplified diagram from their study is presented in Figure 3. The nodes were built based on the understanding of life-cycle processes, from requirements specification through testing. The probabilities of respective states were based on the analysis of literature and common-sense assumptions about the relations between variables. The node variables are shown on histograms of the predictions obtained by execution of the network after the evidence entered (the evidence is represented by nodes with probabilities equal to 1.0). As the authors say, the advantage of their model is that it "provides a way of simulating different events and identifying optimum courses of action based on uncertain knowledge."

Software Dependability. Delic et al. [4] used BBNs to formalize reasoning about software dependability to facilitate the software assessment process. They constructed a network for evaluating dependability of a software-based safety system. It used the data associated with two primary assumptions: the excellence in development (called a process argument) and failure-free statistical testing (called a product argument). The network topology is presented in Figure 4 only to illustrate the factors taken into consideration. The actual probability distributions have been derived from a sample of programs from an academic experiment and are not shown on the diagram.

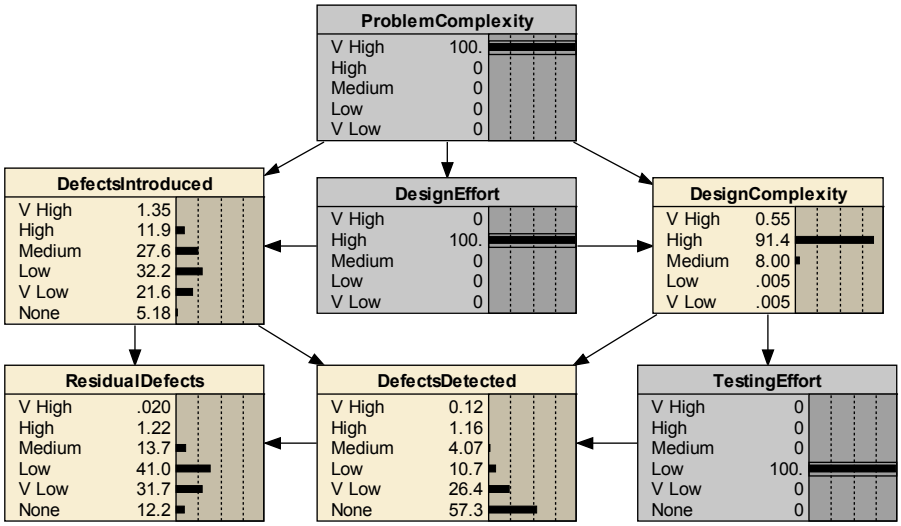

Fig. 3. A simplified BBN model for assessing defect density (Neil and Fenton, 1996)

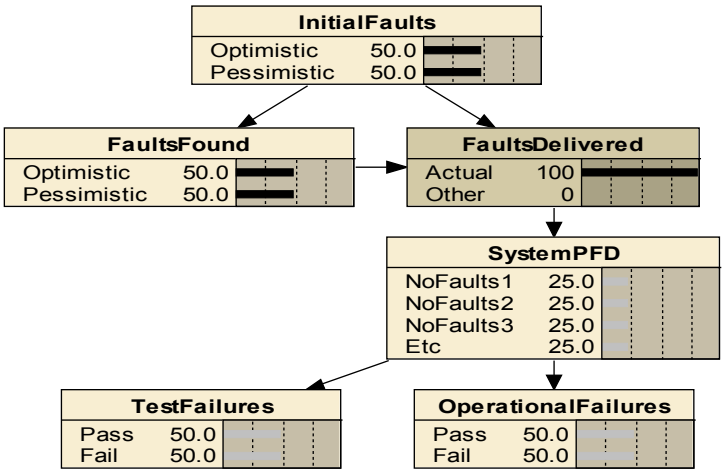

Fig. 4. BBN model for assessing software dependability (Delic et al, 1998)

The authors were interested in estimating the probabilities of failure during acceptance testing and during the operational life of the product (two bottom leaf nodes in Figure 4), given the prior probabilities and observed events. In particular, positive results of an acceptance test allowed deriving numerical estimates about the PFD (probability of failure per demand) and operational performance of the product.

Software Safety. Dahl and Gran [3] applied BBN's to address safety assessment of software for acceptance purposes, in a more comprehensive way, using multiple information sources, such as complexity, testing, user experience, system quality, etc. Their BBN network for system quality, which is only a part of the entire model, is shown in Figure 5. It involves two root nodes: *UserExperience* and *VendorQuality*, and a number of leaf nodes, corresponding to observable variables, of which *QualityMeasures* is of particular importance. This node shows evidence about the system quality, grouping quality attributes, such as readability, structuredness, etc., and can be expanded further.

Other observable variables include *FailuresInOther Products*, those related to the user experience (*NoOfProducts* and *TotalUseTime*), as well as those related to quality assurance policy. When evidence becomes available, entering respective observation data into these nodes and executing the network provides assessment of the variable in question, which in this case is *SystemQuality*.

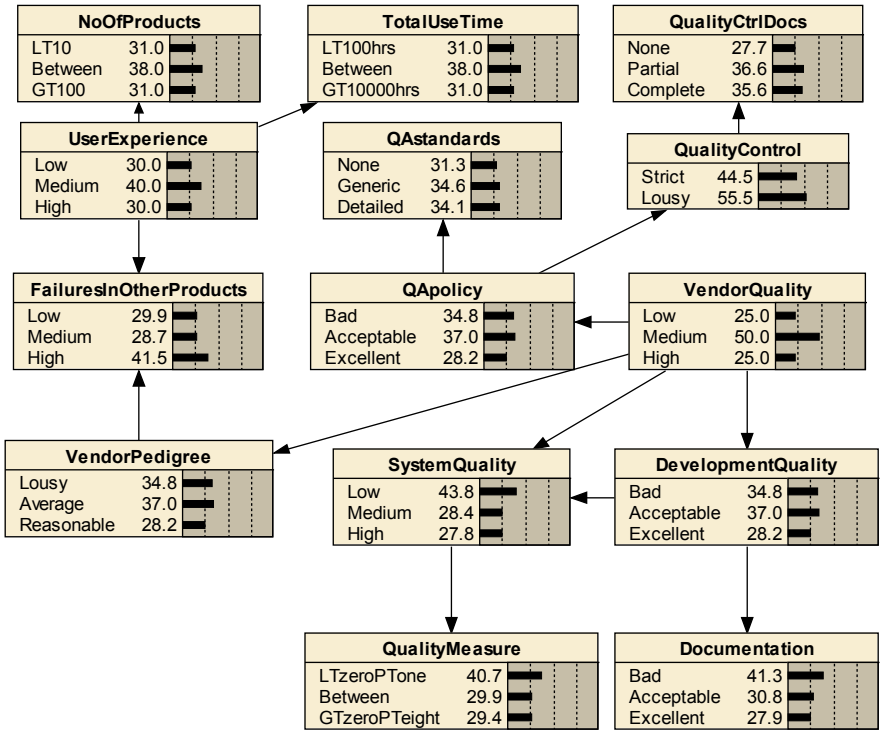

Fig. 5. BBN for system quality parameter in safety assessment (Dahl and Gran, 2000)

The authors note, however, that their example is intended more as an illustration of the method rather than as a real attempt to compute the quality of the system. Their probability assignments to the node variables were chosen somewhat *ad hoc*, and not based on any deeper analysis of the problem. However, as the authors say in conclusion, the results of the study were positive and showed "that the method reflects the way of an assessor's thinking during the assessment process."

Software Reliability . Helminen [6] used BBN's to approach the problem of software reliability estimation. His primary motivation to apply BBN's was that they allow all possible evidence (large number of variables, different potential sources, etc.) to be used in the analysis of the reliability of a programmable safety-critical system. The essential characteristic of such systems is that they involve a significant number of variables related to reliability, with very limited evidence.

The reliability of such systems is modeled as a *probability of failure* , that is, the probability that the programmable system fails when it is required to operate correctly. To develop an estimate of probability of failure, the authors built a series of BBN models, using evidence from such sources, as the system development process, system design features, and pre-testing, before the system is deployed. This is later enhanced by data from testing and operational experience.

The essential part of this work was building BBN models for various operational profiles for multiple test cycles, involving continuous probability distributions. As a result, using BUGS software that combines Bayesian inference with Gibbs sampling (BUGS, 1996), via Markov chain Monte Carlo (MCMC) simulation, it was possible to estimate, how many tests had to be run for a single system in a particular operational environment to achieve certain level of reliability. To decrease the huge number of necessary tests, multiple operational profiles for the same system were used, which required building replicated BBN models to include other profiles' evidence. In essence, by expanding the BBN models further, this approach also allows reliability estimation over the entire lifespan of the software product, but respective experiments have not been conducted in this study.

3 Software Development Tools Assessment

3.1 Basic Concept of Software Tool Assessment

Software development tools are defined as programs used to automate development of the application software. Such tools are widely used in the development of software in a variety of application domains. In regulated industries (aviation/aerospace, nuclear, medical, transportation) there is obvious concern that the selected tool may insert errors into the software it produces. The tools need therefore to be cautiously evaluated for their use.

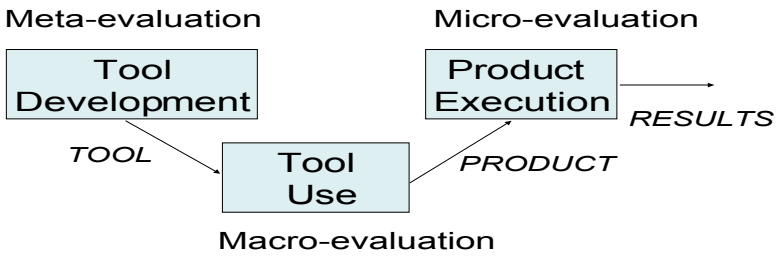

Fig. 6. Model for software tool evaluation process

In an earlier work [9], a rationale has been proposed, based on a model of the tool evaluation process that allows the analyst to select specific software development tools technical evaluation criteria. The framework for this process, derived from the context of tools use, is shown in Figure 6. The central part of this model is the *macro-evaluation* based on the use of the tool during the design phase. However, significant information on tool quality can be derived from the development of the tool itself, considered as a *meta-evaluation*: evaluating the process to develop a tool. The tool vendor can provide the data for evaluation of this stage. In addition to the *macro-* and *meta-evaluation*, the product developed with a particular tool can be included in the evaluation. This is called *micro-evaluation* and focuses on the level lower than the tool itself. Such product evaluation can be based both on static code analysis and on code execution. Consequently, to have the entire picture of the tool's quality, we need to do the evaluation at three different levels.

Only criteria relevant to the design process were included, as opposed to those spanning the entire development cycle, such as maintainability, modifiability, portability, reusability, etc. Two specific criteria, from the ample list of those taken into consideration, are particularly relevant in this regard: *efficiency* of the generated code, which allows conducting forward evaluation regarding the quality of code, and *traceability*, which allows backward evaluation regarding the tool capability of maintaining the right requirements. In addition, it is necessary to evaluate the tool during its operation from perspective of the functions it provides and the ease of use. Two criteria that seem to best capture this operational tool use are: *functionality* and *usability*.

It is important to note that two essential tool evaluation criteria, *reliability* and *robustness*, were not used, for a variety of reasons. Currently accepted reliability measures are based on statistical data and collecting them, even for a single tool, would require a lengthy study, much beyond the time frame of the project. To evaluate tool robustness properly, one needs to apply a wide range of input data to the tool, which was not possible in this research, due to resource limitations.

For criteria selected that way, a series of experiments were conducted, with six industry-strength tools applied to the embedded software development. The respective criteria were quantified using the following measures:

- *Usability* measured as development effort (in hours)
- *Functionality* measured via the questionnaire (on a 0-5 points scale)
- *Efficiency* measured as code size (in LOC)
- *Traceability* measured by manual tracking (in number of defects).

Data for some criteria were collected in multiple aspects, for example, data involving the development effort were divided into four categories: preparation, modeling and code generation, measurements, and postmortem (including report writing). Data related to the traceability from requirements to design to code were also collected. Details of the software requirements and actual experiment results are discussed in [9].

3.2 Application of BBN's to Software Tool Assessment

Based on the model of the tool evaluation process, as shown in Figure 6, and the results of experiments with the selected evaluation criteria, as outlined in the previous section, our high-level model of a BBN for tool assessment is illustrated in Figure 7.

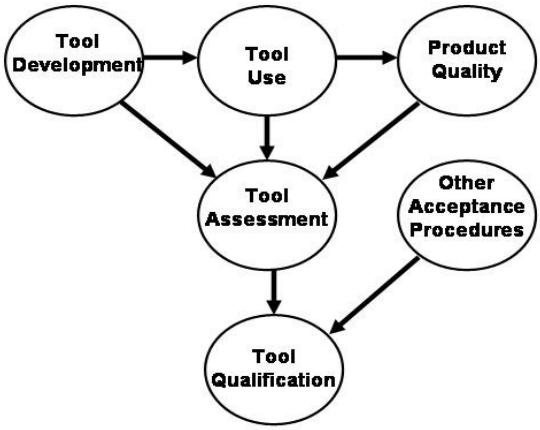

Fig. 7. High-level BBN model for software tool evaluation

Consistent with Figure 6, its primary assumptions are that the tool assessment process should involve the following mutually interrelated factors:

- Development of the tool itself (including the process, vendor quality and reputation, their quality assessment procedures, etc.)
- The tool use (including experimental evaluation based on predefined criteria, but also previous user experiences with this tool, etc.)
- Quality of the products developed with this tool, based on product execution, static code analysis, etc.

Based on the results of this analysis and other acceptance procedures (such as, legal aspects, independent experts opinions, etc.), the tool qualification process can be completed, as depicted in a BBN in Figure 7.

In this study, because of the available data obtained from experiments, only the part of the model related to tool use is covered. The logic of the BBN is similar to the ones reported by Dahl and Gran, where they had no real probability data [3], or where the conditional probability values "were estimated based on judgments in a brainstorming activity among the project participants" [5].

Experimental data were collected for six tools, coded as tools L, M, N, O, P and Q. Figure 8 present an example BBN assessment for a tool, which is likely to pass the qualification process (80% of high and high/medium). An example for the tool which is not likely to pass qualification (75.6% medium and below) is shown in Figure 9.

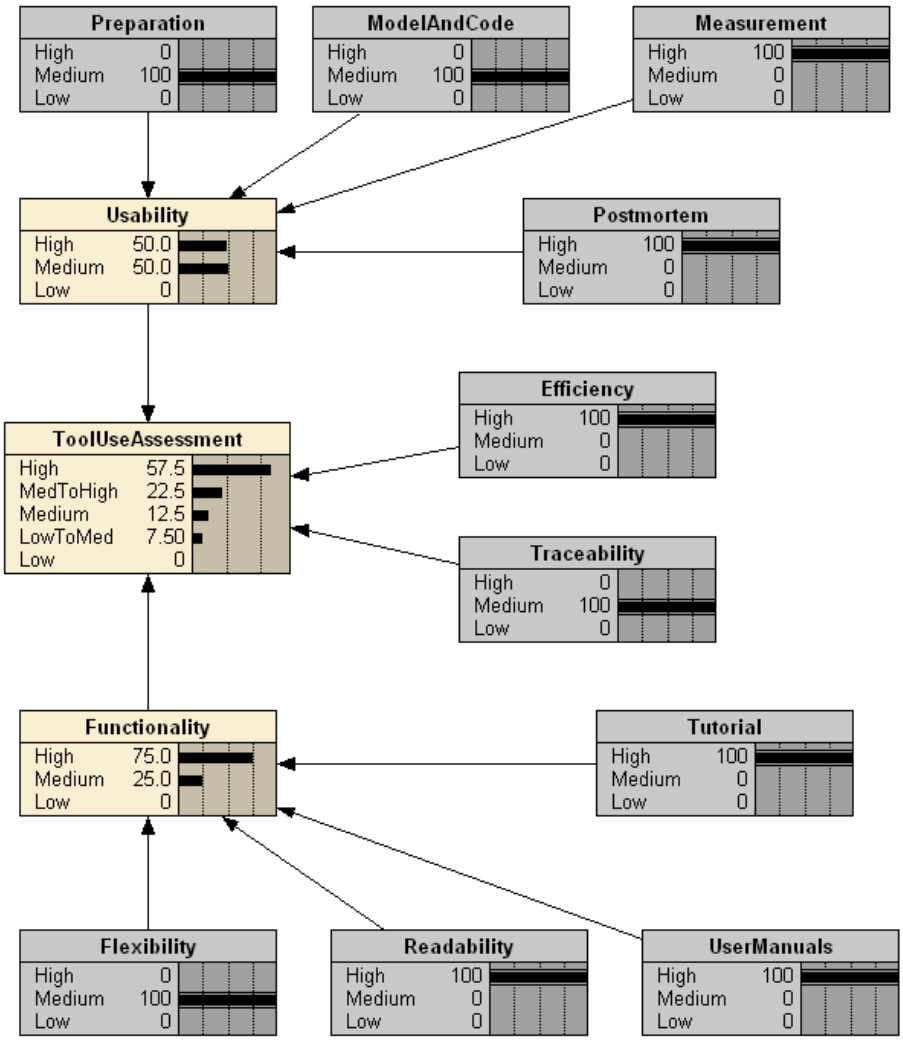

Fig. 8. Example: BBN to assess quality of tool L (80.0% high/medium-to-high)

4 Conclusion

This study proves the applicability of Bayesian belief networks (BBN's) as a support tool for the numerical assessment of software development tools in real-time safety-critical applications. They can be used as a part of the tool qualification process, as well as play a supportive role in making decisions on certification in certain industries.

Acknowledgement

The authors acknowledge the Federal Aviation Administration Aviation Airworthiness Center of Excellence supporting the software tool research under contract DTFA0301C00048 and the support from the Air Force 2006 Summer Faculty Fellowship Program.

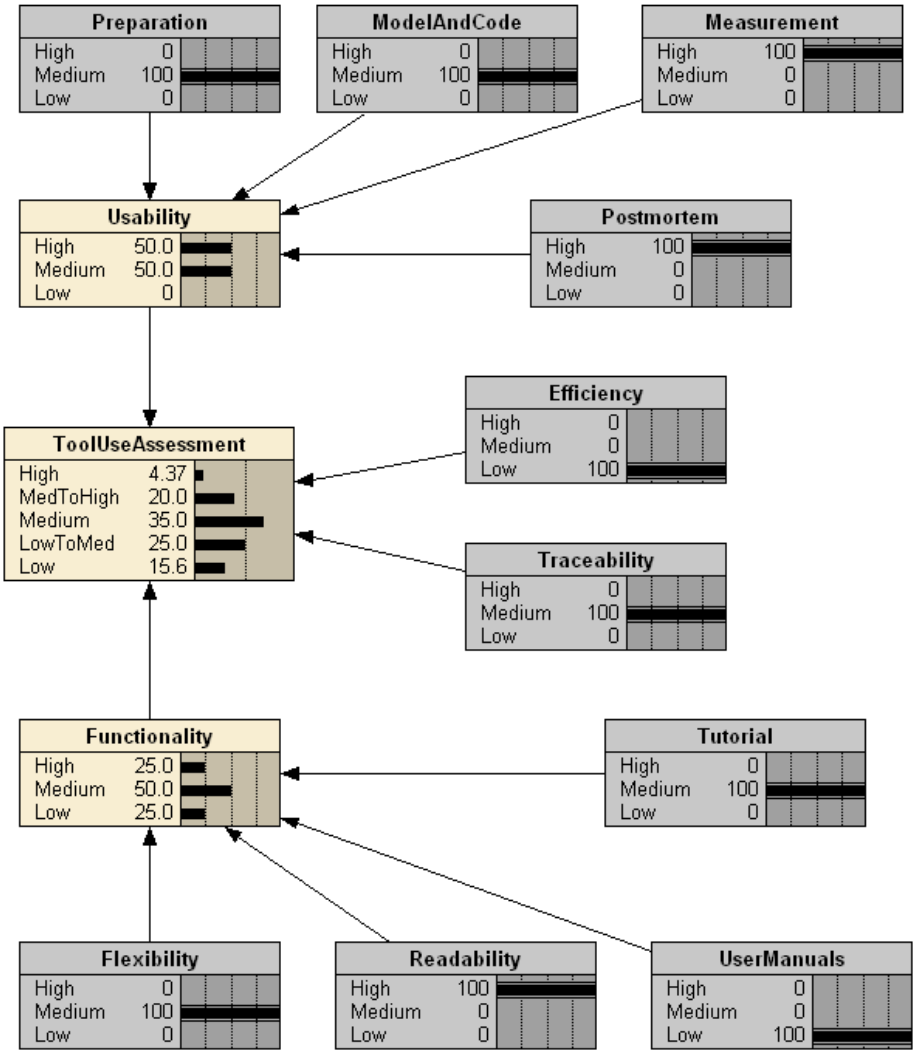

Fig. 9. Example: BBN to assess quality of tool O (75.6% medium and below)

References

1. AgenaRisk (2003), Agena Limited, London, UK, URL: `http://www.agena.co.uk/`
2. BUGS (1996), MRC Biostatistics Unit, Cambridge, UK, `http://www.mrc-bsu.cam.ac.uk/bugs/welcome.shtml`
3. Dahl G., B.A. Gran (2000), *The Use of Bayesian Belief Nets in Safety Assessment of Software Based Systems, Int. Journal of General Systems,* Vol. **29** , No. 2, pp. 205-229.
4. Delic K.A., F. Mazzanti, L. Strigini (1998), *Formalising Engineering Judgement on Software Dependability via Belief Networks,* Proc. DCCA-6, 6th IFIP Int. Working Conf. on Dependable Computing for Critical Applications, M. Dal Cin, C. Meadows, W.H. Sanders, (Eds.), IEEE Computer Society Press, pp. 291-305.
5. Gran B.A., G. Dahl et al. (2000a), *Estimating Dependability of Programmable Systems Using BBNs* , Proc. SAFECOMP 2000, 19th Intern. Conf. on Computer Safety, Reliability and Security, Springer-Verlag, Berlin, pp. 309-320.
6. Helminen A. (2001), *Reliability Estimation of Safety-Critical Software-Base Systems Using Bayesian Networks* , Report STUK-YTO-TR 178, Radiation and Nuclear Safety Authority, Helsinki, June 2001
7. Hugin Explorer (1989), Hugin Expert A/S, Aalborg, Denmark, URL: `http://www.hugin.com/`
8. Jensen F.V. (1996), *An Introduction to Bayesian Networks* , Springer-Verlag, New York.

9. Kornecki A., J. Zalewski (2005), *Experimental Evaluation of Software development Tools for Safety-Critical Real-Time Systems.* NASA Journal on Innovations in Systems and Software Engineering. Vol. **1** , No. 2, pp. 176-188, September 2005.
10. Kornecki A., J. Zalewski (2006), *The Qualification of Software Development Tools From the DO-178B Certification Perspective.* Crosstalk – The Journal of Defense Software Engineering, Vol. **19** , No. 4, pp. 19-23, April 2006.
11. Neil M., N. Fenton (1996), *Predicting Software Quality Using Bayesian Belief Networks,* Proc. SEW-21, Ann. NASA Goddard Software Engineering Workshop, December 4-5, 1996. `http://sel.gsfc.nasa.gov/website/sew/1996/topics/neil_p.pdf`
12. Netica (1995), Norsys Software Corporation, Vancouver, Canada, URL: `http://www.norsys.com/`