ESTIMATING SOFTWARE RELIABILITY FROM PROCESS AND PRODUCT EVIDENCE

Ganesh J. Pai[1], Susan K. Donohue[2], and Joanne Bechta Dugan[1]

[1]Department of Electrical and Computer Engineering
[2] Department of Systems and Information Engineering
The University of Virginia, Charlottesville, VA 22904, USA
{gpai, skd9f, jbd}@virginia.edu

ABSTRACT

We present a BBN-based methodology to provide an initial estimate of software reliability using expert beliefs and qualitative information on the processes employed in creating the software in this paper. We illustrate our methodology using a simple example of a digital weight monitoring system. We develop an estimate of the reliability of the system's software component using a BBN. We propagate these confidence levels through a fault tree, and observe the results of fault tree analysis (FTA). We observe that a combination of the two methodologies enhances FTA and reliability analysis in general.

KEYWORDS

Probabilistic risk assessment, software reliability, Bayesian belief networks, fault tree analysis, reliability modeling and analysis, sensitivity and uncertainty analysis

INTRODUCTION

Traditional techniques for estimating or predicting software reliability include the use of reliability growth models and architecture-based models; see, for example, Lyu (1996) and Gokhale, et al (1998). These techniques have been accepted as ones that work fairly well when data is available for the software being analyzed. However, obtaining a realistic first pass estimate of software reliability when no data is available still remains an open problem, and it's a problem that these techniques do not address. The problem is compounded when we also consider that the task of determining software reliability involves uncertainties. Further, none of these models consider the influence of engineering judgment, qualitative analyses, or the influence of software standards that are frequently used in software engineering projects. We propose using Bayesian belief networks (BBNs) and fault trees (FT) to model a solution to this problem.

Bayesian belief networks, also known as belief net (work)s, causal net (work)s, probabilistic cause-effect models, and probabilistic influence diagrams, are graphical "expert systems" that incorporate subjective and conditional probabilities. BBNs are uniquely suited to the modeling and processing of both qualitative and quantitative data and evidence from disparate sources, and are an accepted method of evaluating a situation when the outcomes are unknown. A BBN is a special case of an influence diagram.

A BBN is a directed acyclic graph (DAG) composed of nodes and arcs. The nodes represent events or random variables; the arcs show causal relationships or dependencies. Each node has a node probability table (NPT). The NPTs of root or parent nodes contain the probability of a given state for that event occurring; the NPTs of the other nodes contain the probability of a given state occurring conditional on the state of prior, causally linked events.

Fault trees are versatile mathematical models that graphically represent the logical relationship between basic failure events and system failure. Given the failure probabilities of the basic events, fault tree analysis (FTA) can determine the probability of system failure. Fault trees can be used to estimate software reliability by representing the software architecture as a fault tree, in which software components are basic events with failure probabilities connected in a logical path to failure.

Our approach is to use a BBN to incorporate evidence and belief about the software's observable and posited failure states. The resulting failure probabilities can then be mapped into the corresponding basic event involved in the system-level FTA. The idea is to use the BBN to develop a model that permits us to obtain an initial estimate of failure probability for software components, and then refine the estimate as evidence is obtained. Further, the model accounts for uncertainties in reliability estimation and also considers the influence of engineering judgment, qualitative analyses and expert opinion. We use Galileo (Dugan et al, 2000), a widely accepted tool for fault tree analysis, in our work.

OUR PROPOSED METHODOLOGY

Our proposed methodology starts with the estimation of failure rates for basic events for which little or no quantitative data as to reliability exist, or for which qualitative data, such as engineering judgment or expert opinion, as to reliability exist. Examples of these basic events include software components, software-hardware interfaces, and human-influenced failure events; our example will focus on a software component. We will use a BBN to estimate the failure rate of the software component.

We then develop a BBN to provide probabilities that a basic event's failure rate belongs to a given range of values. For example, the states of the root or target node of a BBN would be "probability that the failure rate is…" greater than .95, from .95 to .9, and less than .9.

We would then populate the node probability tables and obtain the initial, or *a priori*, probabilities, for the nodes' states. We would gather evidence as to the actual state for as many nodes as possible; the updated probabilities determined by the instantiation of states are referred to as posterior probabilities. Sources for both the initial entries in the node probability tables and later evidence include test results, expert opinions, reliability data from similar or earlier versions of the software, and project documentation.

At this point in the development of the methodology, we will use a point estimate of the failure rate in our example. If the most likely state represents a range of failure rates, then we use the midpoint of the range as the basic event's failure rate. If the most likely state is an open interval, then we use the value at the closed end of the interval as the basic event's failure rate.

We would then populate the basic event (BE) in a FT with the assessment and evaluation results from the BBN, and propagate the results through the FT.

We describe the methodology in greater detail in the next section using a simple hypothetical example system with a software component.

EXAMPLE/CASE STUDY

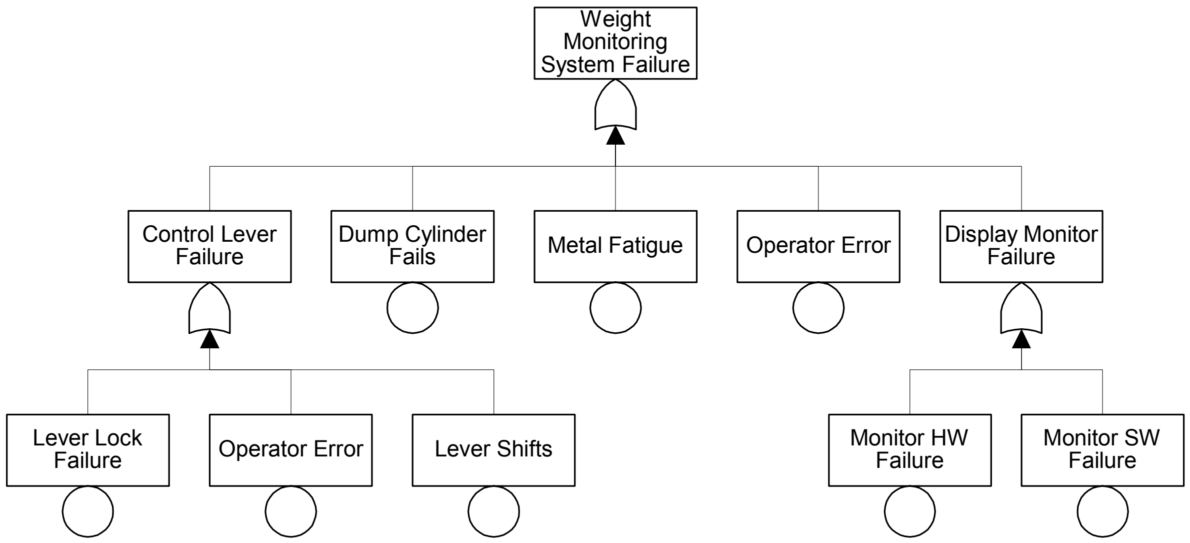

Figure 1: Fault tree for failure of example digital weight monitor system

We consider a very simple system that reports the weight of the load in a front-end loader bucket to the loader's operator in our example. Let's assume that the digital weight control system on a front-end wheel loader must be a reliable system, and that we need to compose a case for replacing an analog weight monitor system with a digital one. We will thus need to assess the failure rates of given failure events to determine that the digital system is at least as reliable as the analog system before the replacement is approved.

The analog system was composed of an array of sensors would transmit a "too heavy" signal to the operator when the bucket carried an amount of material that weighed more than a pre-determined weight limit. The digital system is composed of an array of sensors and a computer display that reports the weight of material, in pounds, in the bucket to the operator.

The operator, especially an inexperienced one, relies on the system to report the correct weights. A bucket that carries too much weight will eventually fail, thus causing damage to the hydraulic system that operates the bucket. An overloaded bucket is also a destabilizing force, and can cause the loader to tip over, an action that could result in damage to the loader and injury or death to the operator. Figure 1 shows the fault tree for one failure event that must be evaluated: bucket released load prematurely.

We do not have reliability data on the monitor system's software; thus, we will need to develop an estimate of the software's failure rate. We will use a BBN to accomplish this task.

Estimating Software Reliability Using a BBN

Figure 2 is an illustration of the BBN[1] used to compute the estimate of the software's reliability; the belief-bar representation is shown. The BBN we use incorporates the items we have determined,

[1] Our BBN software is Netica; see www.norsys.com for more information.

through research, that need to be evaluated when estimating reliability based on engineering judgment. It is one way to model the assessment process; we acknowledge that there are other models that could be used.

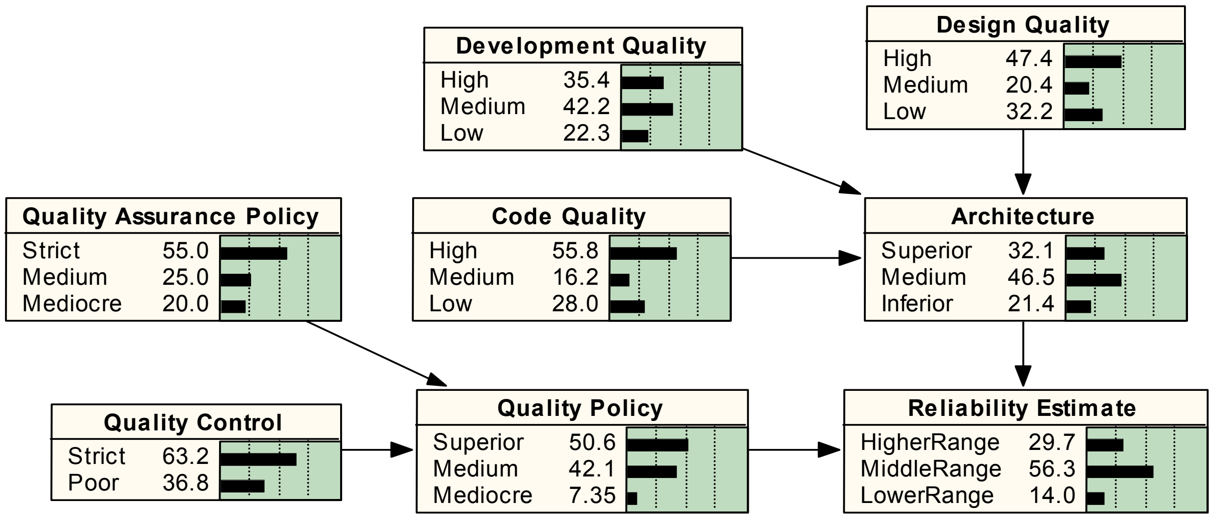

Figure 2: BBN to estimate software failure rates (*a priori* probabilities listed)

The nodes *Quality Assurance Policy, Quality Control, Development Quality, Code Quality* and *Design Quality* represent the factors that influence software failures directly. The nodes are assigned state values (probabilities) based on the history of the organization developing the software. The transitions between the nodes are directed and represent weighted causal relationships.

We used hypothetical values for the node states and transition biases. The weights are assigned on the basis of qualitative analysis, which is typically comprised of engineering judgment and/or expert beliefs.

The values of the states of the *Reliability Estimate* node represent the belief that the reliability estimate lies in the range {> 0.95, between 0.9 and 0.95, > 0.9}. The initial or *a priori* probabilities for the states of the Reliability Estimate node are listed in the above figure; we have a 14% probability that the reliability estimate is less than 90%, a 56.3% probability that the reliability estimate is between 0.9 and 0.95, and a 29.7% probability that the reliability estimate is greater than 0.95. Given this information, we can make an estimate of what the reliability for the software component might be and use it to compute system unreliability via a FT. We will use a point estimate for the failure probability: 1%, or the lower bound of the range with the highest probability.

The values of the states of the *Reliability Estimate* node are updated and refined as evidence is obtained regarding the values of the states of each of the nodes. For example, if evidence of a strict quality assurance policy were obtained, the 3-tuple for the node *Quality Assurance Policy <strict, medium, mediocre = 55, 25, 20>* would change to *<strict, medium, mediocre = 100, 0, 0>*. The resultant probabilities are called posterior probabilities. We will use both the initial and the posterior probabilities in the analysis.

Analysis and Observations

TABLE 1
RESULTS OF FTA FOR THE WEIGHT MONITORING SYSTEM

Basic Event	Failure Rate (λ)	Failure Probability (P)	System Unreliability (Q)
Lever Lock Failure	0.0045	-	
Operator Error	-	0.03	
Lever Shifts	0.00065	-	*Q = 0.66103*
Metal Fatigue	0.0035	-	
Monitor HW Failure	0.00000565	-	
Monitor SW Failure	-	0.1	

An exponential failure distribution was assumed for all basic events except the software module and operator error. The analysis was performed for a mission time of 100 days. From **Table 1**, we see that the unreliability computed for our failure event is .66103. The estimate can be refined as evidence is obtained. The BBN thus provides a path to include evidence in reliability analysis. As a consequence, the reliability analyst has greater confidence in the software because the evidence has also been incorporated. It is also important to note that these values are not truly representative but only serve to demonstrate how the methodology can be applied.

TABLE 2
RESULTS OF FTA WITH EVIDENCE OBTAINED FOR QUALITY ASSURANCE POLICY

Node	State Values (Old)	State Values (New)
Quality Assurance Policy	<55.0, 25.0, 20.0>	<0.0, *100.0*, 0.0>
Reliability Estimate	<29.7, *56.3*, 14.0>	<32.1, *60.0*, 7.86>
Reliability Estimate (point value)	<99.0, 1.0>	<99.0, 1.0>

Table 2 lists the change in state values for the *Reliability Estimate* node when evidence is provided for the *Quality Assurance Policy* node that the quality assurance policy can indeed be rated as “medium.”

Note that our belief that the estimate of the software’s reliability is between .9 and .95 is enhanced by the evidence; the posterior probability for the state representing that range of values went from .563 to .6. Concomitantly, the posterior probability that the reliability is .95 or higher went from .297 to .321, and the posterior probability that the reliability is .9 or less went from .14 to .0786.

Table 3 shows the results of FTA when point estimates are chosen from the range of values provided by the *Reliability Estimate* node, with the software quality remaining unchanged. Propagation of this range of values adds value to the results of FTA by providing the designers with a range of values within which the unreliability of the system may lie.

TABLE 3
RESULTS OF FTA WITH PROPAGATION OF POINT ESTIMATES FROM CONFIDENCE LEVELS

Node	State Values (Old)	Unreliability (Q)
Reliability Estimate	<29.7, 56.3, 14.0>	
SW Reliability Estimate (Point Values)	<99.999, 0.001>, <99.0, 1.0>, <85.0, 15.0>	0.696057, 0.69907, 0.74163

CONCLUSIONS

Incorporating BBNs into fault tree analysis allows an analyst to perform more accurate analysis of a failure event that is composed of basic events for which little or no quantitative reliability data exist, or for which qualitative reliability data primarily exist. We can develop an estimate of failure rates by blending qualitative data, expert opinion, and/or engineering judgment with quantitative data, or by assessing the qualitative data, expert opinion, and/or engineering judgment on its own by using BBNs.

This paper represents only preliminary work in the open problem of determining software failure probabilities in the absence of quantitative data. We would like to validate our methodology and create a proof-of-concept model incorporating this methodology with the Galileo fault tree analysis tool.

REFERENCES

Joanne Bechta Dugan, Kevin J. Sullivan, and David Coppit. (2000) “Developing a Low-Cost High Quality Software Tool for Dynamic Fault Tree Analysis.” *IEEE Transactions on Reliability* **49(1)**, pp. 49–59.

S. Gokhale, W.E. Wong, K.S. Trivedi, and J. R. Horgan. (1998) “An analytical approach to architecture-based software reliability prediction.” *Proceedings of the International Performance and Dependability Symposium*, pp.13 - 22.

Michael R. Lyu, editor. (1996) *A Handbook of Software Reliability Engineering*. New York, NY: McGraw Hill / IEEE Computer Society Press.