Formalising Engineering Judgement on Software Dependability via Belief Networks

Kemal A. Delic[1], Franco Mazzanti,
IEI-CNR, Pisa, Italy
kemal_delic@HP-France-om4.om.hp.com, mazzanti@iei.pi.cnr.it,
Lorenzo Strigini
Centre for Software Reliability, City University, London, U.K.
strigini@csr.city.ac.uk

Abstract

We present the use of Bayesian belief networks to formalise reasoning about software dependability, so as to make assessments easier to build and to check. Bayesian belief networks include a graphical representation of the structure of a complex argument, and a sound calculus for representing probabilistic information and updating it with new observations. We illustrate the method and show its feasibility via a simple example, developed via a commercial computer tool, representing a form of argument which is often used in claims for high dependability. This example is not meant to be "typical", since a sound and complete argument can only be built using the knowledge available in the specific case of interest. This example, although simple, demonstrates the advantages of using belief networks for sounder assessment of reliability and safety.

1. Introduction

The probabilistic assessment of the dependability of software products is a formidable task, for which no proven method is available. This problem is especially serious in the context of safety-critical systems. Accepted standards and guidelines only help in checking that recommended or prescribed practices were applied to ensure dependability, not that they achieved it. Mathematically rigorous conclusions are only possible from very specific types of evidence,

[1] Kemal Delic is now with Hewlett-Packard France.

Proc. DCCA-6, 6-th IFIP International Working Conference on Dependable Computing for Critical Applications, "Can We Rely on Computers?", Garmisch-Partenkirchen, Germany, March 1997.

e.g., results of statistical testing, and are often insufficient for certifying the required levels of reliability or safety [12, 19].

The assessor (developer, independent assessor or licensing authority) is confronted with a wealth of evidence about the design methods used, the quality assurance organisation, the results of testing, etc., none of which is usually sufficient to prove the desired conclusion, e.g., that a system has a certain small probability of dangerous failure. In these conditions, the assessor uses "engineering judgement" to integrate all evidence into a statement that the software is safe (or reliable) enough. This step of integrating heterogeneous evidence into a single, probabilistic statement is an essential, unavoidable phase in the assessment. No improvement in software engineering (e.g., wider use of formal methods) will eliminate its necessity (for a more complete argument, see [12]).

In this judgement, experts rely on their previous experience as well as on the evidence about the individual project they are assessing. However, they cannot usually articulate how the available evidence is sufficient to support the desired conclusion. The net of cause-and-effect chains, deductions and inferences which binds the evidence with the conclusions is presumably rational, but too complex for analytical description. However, such informal judgement in complex inference is often untrustworthy [20]. We have looked for explicit, formal ways of describing the reasoning that leads to an assessment of software reliability. The benefits we expect from such "formalisation" are that assessors could double-check their own reasoning for the consistency and plausibility of its premises and deductions, and the assessment would be open for inspection by other assessors, corporate decision makers, licensing authorities. The consistency of the arguments can be checked by automatic tools, so that the human assessors can concentrate on the correctness of its premises and the structure of the reasoning, rather than on the complex computations involved.

The problem we are discussing is beyond the scope of application of established methods like fault tree analysis. A fault tree gives a convenient representation of *deterministic* cause-effect relationships among events; probabilities can then be assigned to causes and propagated to effects. Assessing software dependability (or the dependability of any other system where complexity and design faults are major concerns) requires one to consider causal relationships that are *probabilistic* in nature. For instance, using a given design verification method is likely to improve the reliability of products *on average*, yet wide variations are possible among products to which the method has been applied. Some causes of this variation may be known (e.g., the skill of the people who apply the method), yet difficult to measure, and a degree of random variability remains even between products developed under virtually identical conditions.

Even when modest levels of dependability are concerned, having an argument with a trustworthy logical structure would be an improvement (in terms of better confidence in the conclusions) over the current state in which quantitative assessment, if provided, is usually given as expert opinion, without the possibility of verifying the reasoning which led from the existing evidence to that opinion.

For formalising the judgement process, we chose "belief networks" (also

known as "Bayesian causal networks" and other names). This formalism, with software tools allowing its use by non-mathematicians, is being increasingly applied to decision problems in different fields. It is based on the calculus of probabilities, which is appropriate not only because the goal is probabilistic assessment, but also because most of the reasoning in engineering judgement must be (at a conscious or unconscious level) of a statistical or probabilistic nature: it deals with predicting the outcome of an individual case on the basis of experience with a class of similar cases.

We have studied several examples of belief networks for representing reasoning about software dependability, to evaluate how suitable this formalism is in practice for representing the kind of information that is available for assessing software. We present one such exercise, describing a commonly used form of reliability argument.

In section 2 we briefly introduce Bayesian reasoning and belief networks. Sections 3 presents our example scenario and belief network, and the numerical results obtained by modelling different assumptions are discussed in Section 4. We thus show some of the issues that were made explicit in defining the belief network, the difficulties of the task and the results to be expected. Section 5 contains our conclusions.

2. Belief networks

In recent years, much attention has been directed at graphical models of probabilistic reasoning, by researchers in such fields as artificial intelligence, statistics, medicine, decision analysis and in a wide variety of applications [5, 7, 10, 17, 18]. Software products have been built (e.g., HUGIN [1], DEMOS [8]) to support the use of these notations. What follows is a very brief introduction to the topic; see e.g. [9] for a more complete introduction.

A belief network is a directed acyclic graph, like the one in Fig. 2, associated with a set of probability values. We can use a belief network to represent our uncertain, probabilistic knowledge about a real-world situation by specifying: i) the topology of the graph, and ii) probability tables associated with its nodes. Each node represents a set of events (a partition on the set of outcomes of an experiment or observation), e.g. the values of a numerical random variable. The events are called the possible "values" or "states" of the node. Arcs represent statistical or probabilistic conditioning of the value of a node on that of other nodes. Each node has a table of probabilities associated with it. If the node has no incoming arcs (root node), this table lists the ["marginal"] probabilities of its possible values; if it has *n* incoming arcs, the table lists the probabilities of its values, *conditional* on each possible *n*-uple of values of its "parent" nodes. This information represents the fact that knowledge about a (parent) node is useful for predictions about another (child) node: through cause-effect relationships (e.g., "amount of debugging" and "number of bugs before debugging" would affect "bugs left after debugging"), or via more general correlation laws ("which team developed this software" would affect "number of bugs"). Once one has assigned these probabilities (which is of course a difficult task, and the task where human expertise and critical ability is best spent), an automated tool can:

- calculate the probabilities of events represented by nodes with incoming arcs,

from the tables of conditional and marginal probabilities of their ancestor nodes;

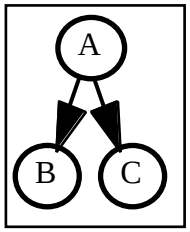
- when an event (value of a node) is actually observed, update the ("prior") probabilities given by the user to other events in the table (by repeatedly applying Bayes' theorem to "propagate" the new knowledge along the arcs in the graph), obtaining "posterior" probabilities that take into account the events observed. The Bayesian inference rule for the posterior probability of a conjecture C in a universe U of mutually exclusive conjectures, on the basis of newly observed evidence E, is

$$Pposterior(C|E) = \frac{Pprior(C)\ P(E|C)}{\sum_{X\ U} Pprior(X)P(E|X)}$$

and software tools supporting Bayesian networks have been made possible by recent, efficient algorithms for applying this rule repetitively through a large network.

A belief network models one's (uncertain) knowledge about a certain situation, and also the arguments that can be built to support a thesis about the probabilities of events in the belief network itself, on the basis of the probabilities of other events. An informal judgement can be formalised into a belief network, in that one can specify a series of links of the form "the truth of statement A would support my belief in statement B", and can also specify *how much* the truth of A strengthens this belief in B, compared e.g. to how much some other truth C would weaken it.

Assigning the prior probabilities is obviously difficult, and this is probably the main obstacle to a wider application of Bayesian methods. Each probability table can be derived either from statistical inference from observed data, or from a probabilistic model of the real-world phenomena described by the belief network, or from expert judgement (with the attendant risks). Choosing a topology - the nodes and the arcs to be included in a belief network - is intuitively easier. However, it is equivalent to stating precise probabilistic assumptions, namely about *conditionalindependence* relations among events. For instance, the net structure shown here on the right means that the state of A determines the probabilities of the states of B and of C. The states of B and those of C are independent, conditional on the state of A (this information is actually carried by the *absence* of an arc between B and C).



3. Application to a software assessment problem

We now demonstrate the use of belief networks via an example: a belief network is used to support reliability claims for a software-based safety system. We do not present this belief network as a "typical" or "general" structure for a formalised "software safety case". No universally appropriate structure exists: a sound and useful argument can only be built using the knowledge that is available in the specific case of interest, and especially about previous experience of the development organisation. Our example is realistic, in that it refers to the information that could be available in a software development

organisation, and describes a structure of argument which is often invoked in reasoning about software reliability [15]. However, this is a fictional scenario, built, like a character in a novel, out of realistic fragments. We chose a simple scenario, and give only a high-level description of it, for reasons of space.

We derived our prior distributions from a sample of programs from an academic experiment. We can only give few details of how we derived these prior probabilities, for reasons of space. In any case, agreeing on acceptable inference procedures would be an important part of negotiating a safety case. The chosen procedure would depend on the knowledge available to the assessors, on any required degree of conservative bias, and especially on the sensitivity of the results of interest to the choice of procedures.

Our scenario

In our example, a dependability claim is based (as is common) on the two supporting arguments of excellence in development ("process" argument) and of failure-free statistical testing ("product" argument). We assume an organisation with a stable development process, and accumulated data about previous projects similar to the one for which the current assessment is performed. Knowledge about previous products allows weaker predictions than would be allowed by comparable knowledge about the individual product under examination, but is still valuable. The product is a protection system, and the desired conclusion is that its probability of failure per demand (Pfd) is less than 10^{-4}. Statistical independence between successive demands (which are distant in time and separated by a complete restart) can be assumed, which greatly simplifies inference from test results. The stylised life-cycle chart below represents the phases in the development process about which useful data are available.

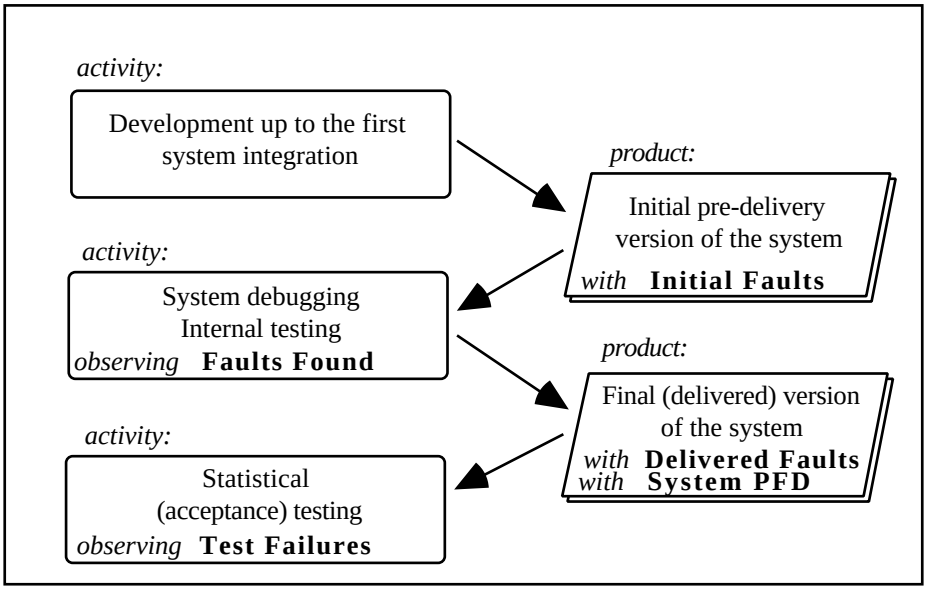


Fig. 1. Life-cycle phases of our hypothetical product.

The argument is then built as follows: by the "process" argument, we claim that the product probably contains few faults, and these are not very serious, which allows us to expect a small Pfd. A final phase of (statistical) acceptance testing (showing no failures in a fixed number of runs, mandated by contract or regulation) then corroborates this belief.

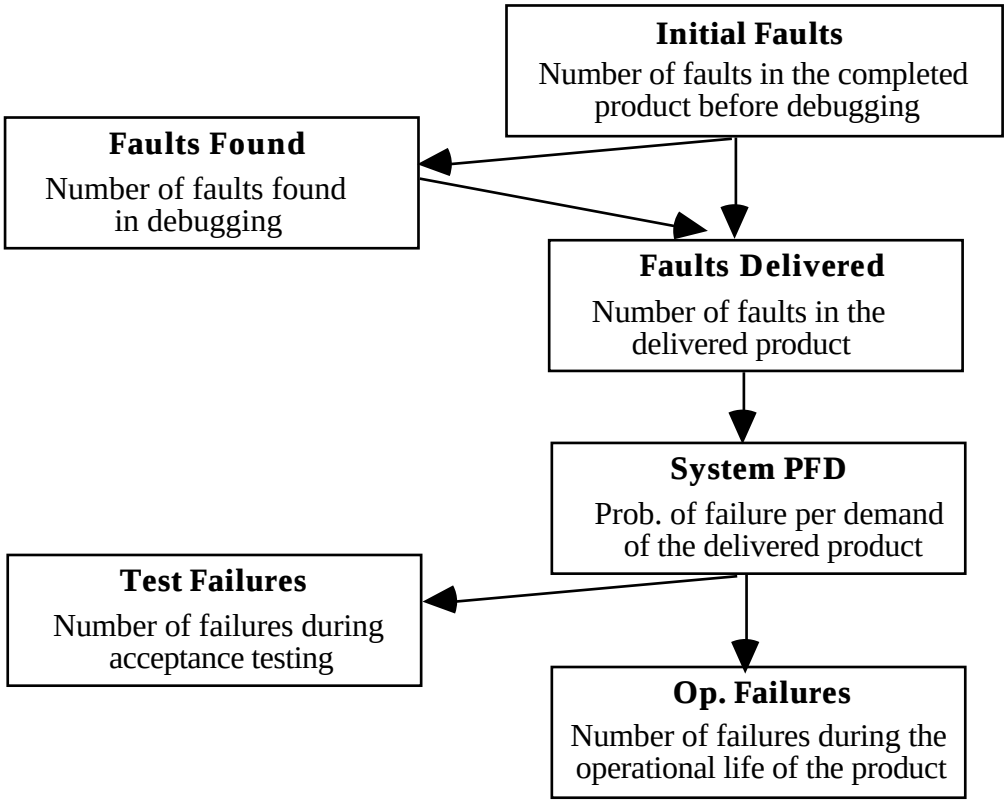


Fig. 2. The belief network for our example.
(Notice the uniform treatment of variables describing past and future events)

The detailed belief net topology is shown in Fig. 2. The process-related evidence (e.g., the skill of the development staff) is not explicitly represented by nodes and arcs. It is represented, though, by assigning the prior probabilities (for those variables that are represented explicitly) on the basis of the distributions observed, for these variables, in the population of the previous projects sharing the same process. If an organisation had a sufficient database of project metrics to allow refined predictions of defect counts, this could be connected to the network shown here, in lieu of the root node **Initial Faults**. We have studied some belief network structures for predicting defect numbers, and others have been published in [16]. So, we have:

- on the one hand the individual project (process and product) being assessed;
- on the other hand, a *population* of previous, similar projects, of which this project can be considered a new sample. They are results of a similar process applied within the same organisation to developing similar products - similar

enough that experience from them should be helpful in making predictions about the new project. "Population-related" data are thus derived from the historical records of the development organisation.

So, knowledge about the process allows some predictions ("prior" beliefs) about the number of faults in the delivered product and their characteristics, and thus about the failure rate of the software. These predictions, available at the beginning of the project, are then refined on the basis of measurements during development, and finally on the basis of statistical testing. This "refinement" amounts to accounting for the way this individual project seems to differ from "the average project" in the population. The relative weighting, in predictions about an individual, of extensive but generic evidence about the population, against limited but specific evidence about the individual itself, is a central issue in process-based arguments, and we shall see that the language of belief networks allows one to state assumptions in this area in an explicit, rigorous fashion and explore their consequences.

The belief network in detail

The node "**Initial Faults**" (in the completed product before debugging) has a prior distribution based on measurements on the population. A possible prior distribution for this node is a binomial distribution:

$$P(X=x)=\binom{M}{x}p^{x}(1-p)^{M-x}$$

This models, for instance, a process in which the developers face a number of likely pitfalls or hurdles, at each of which they might fail and insert a fault into the product. This assumption can be criticised, and it would be reasonable to ask for alternative assumptions to be tested in the belief net. When a prior distribution is inferred from historical data, there will be uncertainty about the appropriate inference procedure and the quality of the data. So, for every such prior distribution, we checked the sensitivity of the results obtained from the belief network to errors in the prior distribution. For this node, we did this by making the mean of the distribution vary between an "optimistic" and a "pessimistic" value, obtained under the condition that they pass a chi-square test ($\alpha = 0.05$) for the population-related data set (informally, this indicates that the data set is reasonably "representative" of a binomial distribution with the hypothesised mean). We have verified that the behaviour of the network depends mostly on the mean of this distribution, and is only marginally affected by the particular values chosen for P and M (whose product is the mean). As a limit case, even a Poisson distribution with the same mean produces similar results.

The node "**Faults Found**", represents the number of faults found during debugging and thus describes the efficacy of debugging. We show the effects of assuming that each fault has the same probability of being found, and that the probability of finding a given fault is not affected by having found another fault. This leads again to a binomial distribution, with M equal to the value of "Initial faults". The other parameter, *p*, is set equal to the observed average over the data set (0.75), and by way of comparison, to the more pessimistic estimate that

on average two thirds of the faults are found.

The value of the node "**Faults Delivered**" is the difference between those of "Initial Faults" and of "Faults Found". So, this is a "deterministic" node, and its probability table is a diagonal matrix. For instance, if "Initial Faults" is 5 and "Faults Found" is 3, the probabilities for "Faults Delivered" are: 1 for the value 2, and 0 for all other values.

The node "**System PFD**" represents the probability of failure per demand for a system which has a given number of (delivered) faults. To derive the Pfd from the number of faults, one needs information about the "size" of each fault, i.e., the probability that it will cause a failure (per demand).

We have used a model akin to the Littlewood model [11]: the fault sizes are identically distributed, independent random variables, as though the developers "drew" faults from a population of available faults. The system Pfd, given that the value of "Faults delivered" is N, is the sum of N independent random variables from this distribution. We are thus assuming that failures due to different faults are mutually exclusive, or, with some approximation, low-probability, weakly correlated events.

We estimated the distribution of the fault size from a collection of operational failure data, assuming a bell-shaped distribution (there is little difference in the results between using a normal distribution, truncated between 0 and 1, or a beta distribution). We used a chi-square test ($\alpha = 0.05$) between the data and the distribution to choose a range for the mean of the distribution, and for discarding other families of distributions (e.g., a uniform distribution), and to set its variance (by best fit to the data). In table 1, we have labelled the assumption of a mean of fault sizes at the high end of our range, or at low end of it, as "pessimistic", and "optimistic", respectively, for intuitive reasons.

The reader may notice that our belief network does not contain a node representing the fault size. This means that the fault size distribution is fixed, and only enters the argument by being used in calculating the distribution of Pfd for each value of "Faults delivered", filled in the table associated to "System PFD".

Trusting the chosen distribution for the fault sizes is clearly crucial for any conclusion about the Pfd of the software. For reasons of space, we can only represent one way of deriving this distribution. However, this is obviously not the only (or the most) plausible way. It is worthwhile to discuss how alternatives should be considered, in a practical application, to deal with this basic source of uncertainty. The derivation used here assumes that the fault size distribution for this product is the same as that of the faults observed in the operation of previous products. A more complex alternative ([6]) would assume that different products have different distributions of fault sizes, belonging to a same parametric family and differing in the values of a set of parameters; the variability between products is represented via the probability distributions of these parameters. As yet another alternative, additional sources of evidence could be tapped: e.g., if the developer had analysed the faults found in the current product during development, their sizes could be argued to be an indicator of the sizes of the remaining faults. In practice, those methods for deriving a distribution of fault sizes that are plausible in the specific circumstances should be modelled and their results compared. The modelling

may entail additions to the belief net ([6]) , to exploit its automated inference procedure. The desired outcome is a distribution that is considered accurate enough, or conservative enough. This may be a matter of consensus among experts examining the formalised argument. We assume for the rest of this paper that such a consensus has been obtained. However, it is also possible that no distribution is found that is considered sound and that will support the claimed value for the system Pfd (for instance, because the results are very sensitive to minor variations in the inference method). This would mean that the form of argument modelled in Fig. 2, although mathematically consistent, is useless in practice for the particular product being assessed. One must then either seek additional evidence or select a different argument (or conclude that the product is unacceptable, of course). We point out in passing that here lies the advantage of "formalising" judgement: if the argument were only described in intuitive terms, no strong check could be made that it really supports the desired conclusions in these specific circumstances. One could then look for ways of reasoning that do not depend on estimating the fault sizes and defect counts. One such way is the "black box" argument described in [13]. We may also note that many software companies now use defect counts as indicators of reliability. This claim implicitly depends on assumptions on fault sizes, and these companies could use the kind of plausibility test that we just outlined.

The node "**Test Failures**" indicates the possible outcomes of acceptance testing, "no failures" or "one or more failures". Their probabilities, in N tests, conditional on "System PFD" having a certain value, $\overline{Pfd}$, are computed (given independence among tests) as:

$$P(\text{no failure in N demands} \mid \overline{Pfd}) = (1- \overline{Pfd})^N$$

$$P(\text{at least one failure in N demands} \mid \overline{Pfd}) = 1-P(\text{no failure in N demands} \mid \overline{Pfd}) = 1-(1- \overline{Pfd})^N$$

The knowledge modelled by this belief network would evolve during the project:

- initially, the belief network is populated with probability distributions obtained from population-related data, and yields a "prior" distribution for the Pfd. One can, for instance, use the probability of satisfying the requirement on the Pfd as an indication of project risk at this stage;
- when the project delivers its first completed version of the system, the knowledge available is still as it was initially. Of course, if management had observed significant departures of this project from the normal progress of previous, similar projects, this belief network would no longer be applicable;
- during debugging, the faults found are logged, at the end their number is "input to" the belief network as the observed state of the node "Faults Found", and the other probabilities in the belief net are updated as a result;
- acceptance testing is then performed for N demands. If no failures happen, the state of the node "Test Failures" is set to "no failures", and all the other distributions are updated as a result;
- a safety case should normally be reviewed periodically. A node representing the "Number of failures in a set interval in operation" could be added, with

the same parents, and a probability table derived in the same way, as for "Test Failures", with a different value of N.

Notice also that a requirement in terms of the unobservable random variable "System PFD" may seem unsatisfactory [14]. It may be preferred to require that a failure (i.e., an observable event) be sufficiently improbable over the lifetime of the system. This is easy to deal with by adding the node "**Op. Failures**". We assumed that the acceptance testing simulates 50,000 demands, and that 500 demands are expected over the lifetime of the system. These numbers are realistic for protection software in the nuclear industry: in the case of the software protection system in the U.K. Sizewell B reactor, the plan was about 50,000 tests (for an initial requirement of Pfd = 10^{-4}) [4].

4. Use of the belief network and results

This section discusses some results obtained via this belief network:

- the probability that the requirement "Pfd ≤ 10^{-4}" is satisfied;
- the probability of failures during acceptance testing. This is of obvious interest in project planning: it is the probability of incurring serious extra delays and costs;
- the probability of failures during the operational life of the product. This could be argued to be the only prediction of actual interest, as it represents the actual risk in operating the product.

We consider the values of these probabilities at three different stages in the project:

- at the beginning, when predicting the likely outcomes of the project;
- after system debugging, knowing how many faults have been found;
- after successful acceptance testing.

These two latter predictions, in the form of "what if" thought experiments, are also useful early on in project planning.

The software tool can directly calculate the probabilities of the events of interest, with the given prior probabilities and observed events. An essential part of the analysis is then to check the soundness and robustness of the argument represented. There are several ways of looking for weak points (to correct them, and to increase confidence in the argument when none is found): i) studying the implications of different network structures (which we cannot do for reasons of space), to verify that alternative, plausible ways of reasoning produce consistent results; ii) examining the conclusions that the belief network would yield with different hypothetical values of the observed variables (e.g., different number of faults found in system debugging), to verify that these conclusions are plausible; iii) verifying that the conclusions reached are not overly sensitive to the unavoidable uncertainties about the premises. We only show here some checks of types ii) and iii): we examine the conclusions that our belief net would produce as a consequence of different observed facts, and we vary the values of parameters that represent characteristics of the stochastic processes of interest. These parameter values are derived from statistical observations, and clearly subject to some uncertainty. If the predictions of interest proved highly sensitive

to this perturbation of parameter values, decision-makers should be reluctant to depend on them, and may well require further investigation of the processes that those parameters describe. We could say that these parameters represent "physical" assumptions. Additional assumptions are usually needed to simplify the computations. We shall call these "mathematical assumptions" for brevity, and we have routinely verified that they do not cause appreciable errors in the results.

Throughout this exercise, we used the HUGIN tool [1] for manipulating belief networks.

Results

Table 1 is a concise summary of the effects of using different parameter values in building the prior probabilities. We add a few comments.

As one would expect, all other factors being equal, "optimistic" assumptions regarding "Initial faults" and/or "Faults found" generally bias the results towards more optimistic predictions. However, it is not guaranteed that a set of hypotheses will imply consistently more optimistic predictions than another set. For instance, a higher probability of failure-free testing does **not** unconditionally imply a higher probability of failure-free operation, or a lower probability of a Pfd greater than 10^{-4}: see for instance the six shaded cells in the table. Assumptions about fault sizes have subtle effects: the larger the faults are believed to be, the more pessimistic -usually - the prior distributions for the Pfd, and the other nodes of interest. However, after passing the acceptance test, this effect is reversed. This can be seen for instance in the four cells marked with a double border. Informally, if I expect that any fault will cause failures with a high probability, testing that does not reveal failures will be very likely to imply that no faults are present (a formal description of this kind of argument and a criticism of its applicability is in [2, 3]). The tool automatically tracks such non-intuitive properties of a model.

There are other examples of how choices of prior distributions may have unexpected implications. For instance, finding more faults during debugging usually (with our set of other hypotheses) implies more optimistic predictions. This, in itself, may appear non-intuitive. It simply means that, from past experience, one expects a few faults to be present, to the extent that finding fewer is more an indication of debugging being unusually ineffective than of the product being unusually good. However, this is not true over all the event space for this belief network, and there is at least one setting of the parameters which makes finding one fault "worse" than finding none.

Assumption for the distributions inferred from population data			Prior probabilities of events, %			Probabilities after finding 0 faults, %		Probabilities after finding 7 faults, %	
initial faults	faults found	fault size (hidden)	Test OK	Op OK	Pfd ≥ 10^{-4}	Test OK	Op OK	Test OK	Op OK
opt	pess	pess	72.6	90.4	20.4	71.3	89.9	82.7	94.1
opt	pess	opt	78.3	95.2	13.0	77.3	95.0	86.5	97.1
opt	opt	pess	79.0	92.8	15.4	77.9	92.4	86.9	95.7
opt	opt	opt	83.5	96.4	9.7	82.6	96.2	89.8	97.9
pess	pess	pess	56.3	83.6	33.7	53.2	82.2	70.1	89.5
pess	pess	opt	64.6	91.6	22.2	61.9	90.9	76.3	94.7
pess	opt	pess	65.7	87.6	25.9	62.6	86.4	76.9	92.1
pess	opt	opt	72.6	93.8	16.7	70.0	93.1	81.8	96.1

Assumption for the distributions inferred from population data			Probabilities after finding 0 faults **and** successful testing, %		Probabilities after finding 7 faults **and** successful testing, %	
initial faults	faults found	fault size (hidden)	Op OK	Pfd ≥ 10^{-4}	Op OK	Pfd ≥ 10^{-4}
opt	pess	pess	99.958	0.014	99.977	0.007
opt	pess	opt	99.947	0.012	99.970	0.006
opt	opt	pess	99.969	0.01	99.983	0.005
opt	opt	opt	99.961	0.009	99.978	0.005
pess	pess	pess	99.922	0.028	99.956	0.014
pess	pess	opt	99.901	0.025	99.944	0.013
pess	opt	pess	99.942	0.02	99.968	0.010
pess	opt	opt	99.927	0.018	99.959	0.009

Table 1. Probabilities derived from the belief network of Figure 2 (the table is split in two parts; the row headings are the same for both parts)

Legenda:

opt (pess): the distribution chosen is the more intuitively optimistic (pessimistic) among the two described in the main text

Test OK: the event: "no failure during acceptance testing"

Op OK: the event "no failure during the operational life of the system"

Pfd ≥ 10^{-4}: the event "the product does not satisfy the requirement on its probability of failure per demand"

The results of the acceptance test dramatically changed the predictions about the Pfd and operational performance of the product (last four columns, compared with the previous four). In other words, the evidence lined up before acceptance testing gave limited confidence that the product would satisfy its

requirements, and acceptance testing was essential for confidence in the product. In other circumstances, "process" evidence might give a high confidence of satisfying the requirements, and in particular of successful acceptance testing: then, actually observing the latter would not change the predictions as much.

An interesting comparison is that between the implications of finding 0 faults and of finding 7. Clearly, with our priors finding 7 faults implies a high likelihood of having found all the faults present. But another consideration applies. From our net we can see that finding 0 faults had a prior probability of about 40 %. Finding 7, by contrast, had a prior probability of about 1 in 10,000. Having found 7, we know that we have observed an event which we considered extremely unlikely. This would prompt us to re-examine our assumptions with added caution: it may suggest that they are violated in this specific project, or at least that small variations in how we model this "tail" of the probability distribution of "Faults found" may have large effects on our conclusions.

5. Conclusions

This paper illustrates the use of belief networks for describing a complex argument, criticising it and updating it with the results of new observations. We have shown a reasonably realistic example of their use, which could be part of an actual safety case.

Our treatment of this example has been necessarily terse. In an actual safety case, a full, detailed discussion would be necessary for all the assumptions used, showing that they do represent the available knowledge and are either well-justified or acceptable as they do not introduce dangerous errors in the conclusions.

More importantly, this process of building and studying the network would normally be iterative, as one would not expect to produce the "right" network at the first attempt, but to use the networks one initially proposed as an aid for clarifying one's own thoughts. In successive versions of a belief network, arcs could be added or discarded; nodes that prove useless could be eliminated, others representing variables which, one realises, have some useful predictive power, or allow the validation of the argument through observation, would be added. This would certainly be a painstaking exercise, not unlike some current, cumbersome procedures for documenting informal safety cases; our claim is that belief networks would make safety assessment exercises more trustworthy, not less time-consuming. However, we expect that an organisation can reuse the belief networks produced for a first application as templates for use in later, similar projects, reducing the cost of this approach.

Our provisional conclusion from our work with belief networks is that they are indeed very useful. The iterative process of building and analysing them was useful in clarifying otherwise informal reasoning, discarding weak arguments and warning of counterintuitive implications of seemingly obvious arguments. With the help of this formalism, an assessor can try and answer questions like: "Are my conclusions the right conclusions to draw from my knowledge (data plus informal insight)? Among my assumptions, which are really crucial for my

conclusions? What are the weak links in my argument, and what information do I need to make them stronger?" In the practice of certification and licensing, the use of belief networks should make it much easier to communicate and discuss claims, and to update a safety case over time, all the way from a preliminary feasibility study to revising it with operational experience.

We have argued that belief nets should be exploited to make the process of judgement more explicit, auditable and thus trustworthy: more scientific, in one word. We must also acknowledge that using a software tool, as needed to perform the required complex calculations, may have negative effects. A complex argument is, after description as a belief net, hidden behind a visually intuitive user interface. The precision of the computer calculation may lend spurious authority to unverified, hidden assumptions. Discipline is essential to ensure that belief nets are used as an aid for communication and comprehension rather than an oracle.

The tools we tried, developed as expert system shells, had insufficient capabilities of displaying the probability distributions in a user-friendly manner and of dealing with the small probabilities needed in safety assessment. Further help for practical use would come from tools for automatically producing probability tables, not only via standard inference procedures from sample data but also from probabilistic models of system behaviour (e.g., system failure rates as a function of component failure rates). Last, sensitivity analysis proved extremely time-consuming. Rather than the collection of ad-hoc tools that we developed for this purpose, professional assessors need a user-friendly tool set, relieving them of the bookkeeping chores involved in the exercise.

We believe that investment in such tools will be rewarded by a substantial improvement in the ability of assessors to understand, refine and audit "engineering judgement" on complex safety cases. A collaborative project, SERENE (SafEty and Risk Evaluation using bayesian NEts) is now under way (under the European ESPRIT research programme) to produce a belief network tool set tailored for safety assessment, taking advantage of experience in industrial case studies. Information about the project is available on the World Wide Web, URL: http://www.hugin.dk/serene/.

Acknowledgements

This work was funded in part by the European Commission through the "SHIP", project (Assessment of the Safety of Hazardous Industrial Processes in the Presence of Design Faults), in the framework of the Environment Programme, the "OLOS" Human Capital Mobility network, and the SERENE project (SafEty and Risk Evaluation using bayesian NEts, Contract no: 22187) .

References

[1] S. K. Andersen, K. G. Olesen, F. V. Jensen and F. Jensen, "HUGIN-A Shell for Building Bayesian Belief Universes for Expert Systems", in Proc. 11th International Joint Conference on Artificial Intelligence, Detroit 1989, 1989, pp. 1080-84.

[2] A. Bertolino and L. Strigini, "Using Testability Measures for Dependability Assessment", in Proc. ACM/IEEE 17th International Conference on Software Engineering,

ICSE 17, Seattle, USA, 1995, pp. 61-70.

[3] A. Bertolino and L. Strigini, "Acceptance Criteria for Critical Software Based on Testability Estimates and Test Results", in Proc. SAFECOMP 96, 15th International Conference on Computer Safety, Reliability and Security, Vienna, Austria, 1996, pp. 83-94.

[4] D. B. Boettcher and P. A. Tooley, "High integrity systems in defence in depth at Sizewell B", High Integrity Systems, 1, pp. 199-209, 1994.

[5] CACM, "Real-World Applications of Bayesian Networks", Communication of the ACM, Special Issue, 38, pp. 24-57, 1995.

[6] K. A. Delic, F. Mazzanti and L. Strigini, "Formalising a software safety case via belief networks", SHIP Project Technical Report SHIP/T/046, July, 1995.

[7] P. Hall, J. May, D. Nichol, K. Czachur and B. Kinch, "Integrity Prediction during Software Development", in Proc. Symposium on Safety of Computer Control Systems (SAFECOMP '92), 1992.

[8] M. Henrion, J. S. Breese and E. J. Horvitz, "Decision Analysis and Expert Systems", AI Magazine, 12, pp. 64-91, 1991.

[9] F. V. Jensen, "An introduction to Bayesian networks", UCL Press (Published in North America by SpringerVerlag NewYork Inc.), 1996.

[10] G. D. Kleiter, "Bayesian Diagnosis in Expert Systems", Artificial Intelligence, 54, pp. 1-32, 1992.

[11] B. Littlewood, "Theories of Software Reliability: How good are they and how can they be improved?", IEEE Transactions on Software Engineering, SE-6, pp. 489-500, 1980.

[12] B. Littlewood and L. Strigini, "Validation of Ultra-High Dependability for Software-based Systems", Communications of the ACM, 36, pp. 69-80, 1993.

[13] B. Littlewood and D. Wright, "A Bayesian model that combines disparate evidence for the quantitative assessment of system dependability", in Proc. SAFECOMP '95, Belgirate, Italy, 1995, pp. 173-188.

[14] B. Littlewood and D. Wright, "On a Stopping Rule for the Operational Testing of Safety Critical Software", in Proc. FTCS25 (25th Annual International Symposium on Fault -Tolerant Computing), Pasadena, 1995, pp. 444-451.

[15] G. B. Moutrey, "Equipment and system performance validation - Sizewell B reactor protection system", in Proc. Proceedings of a Forum on Safety Related Systems in Nuclear Applications, London, 1992, pp. 36-40.

[16] M. Neil, B. Littlewood and N. Fenton, "Applying Bayesian Belief Networks to Systems Dependability Assessment", in Proc. Safety Critical Systems: The Convergence of High Tech and Human Factors. Proceedings of the Fourth Safety-Critical Systems symposium, Leeds, U.K., 1996, pp. 71-94.

[17] J. Pearl, "Belief Networks Revisited", Artificial Intelligence, 59, pp. 49-56, 1993.

[18] D. A. Reed, "Treatment of Uncertainty in Structural Damage Assessment", Reliability Engineering and System Safety, 39, pp. 55-64, 1993.

[19] L. Strigini, "Considerations on current research issues in software safety", Reliability Engineering and System Safety, 43, pp. 177-188, 1994.

[20] L. Strigini, "Engineering judgement in reliability and safety and its limits: what can we learn from research in psychology?", SHIP Project Technical Report SHIP/T/030, July, 1994.