SHIP/T046 v1.9 27 September 1995

Formalising a Software Safety Case via Belief Networks

Kemal A. Delic, Franco Mazzanti,

IEI-CNR, Pisa, Italy

Lorenzo Strigini

Centre for Software Reliability, City University, London, U.K.

The SHIP project (ref. EV5V 103) is being carried out with financial support from the EEC in the framework of the Environment Programme, sub-theme: Major Industrial Hazards.

Abstract

Belief Networks (also known as Graphical Probabilistic Networks and with various other names) offer a useful formal language for stating complex arguments in rigorous, yet visually clear terms. They are thus promising candidates for describing the complex, often unclear reasoning that is often implied, but not described, when reasoning about software dependability, in particular when "engineering judgement" comes into play.

We introduce the problem of building a rigorous safety case for software, and argue the merits of belief networks as an aid for building, criticising and perfecting such safety cases. This first report includes a high-level introduction to Belief Networks, and then introduces and discusses a small but realistic example. Our conclusion is that this method has great potential for making safety arguments easier to communicate and check, and in the end more trustworthy.

Date:	27 September 1995
Doc Type:	Technical report
Doc. Id.:	SHIP/T/046
Version:	v1.9
Release Status:	Final
Availability Status:	Public

Produced in: Task 2.3

Contents

1 Introduction

Evaluating in quantitative terms the safety of software products is quite difficult, and as a result the "software safety case"[1] is usually a weak link in the demonstration of safety of any system that includes computers. A lively debate exists about the need, and the very possibility of giving a probabilistic measure of software reliability or safety; yet, if system-level safety requirements are expressed in terms of probabilities, so must logically be the requirements for any safety-critical subsystem or factor, including software. Furthermore, our knowledge about the possibility of future failures can only be probabilistic. The necessity of evaluating software dependability (reliability or safety) in probabilistic terms is argued in detail for instance in [Littlewood and Strigini 1993].

> Although it may be argued [Leveson 1991] that safety and reliability are goals that call for different engineering approaches, they are similar concepts. Reliability measures relate to the probability of failures; safety measures relate to the probability of unsafe failures. The problems in evaluating them are similar. In many applications of software, evaluating safety separately from reliability is impossible, because any software failure may (because of either the nature of the application, or the lack of safeguards in the design of the system) affect the behaviour of the system in arbitrary, potentially unsafe ways. In the rest of this paper we shall usually refer to safety, although it will be clear that the methods used are applicable to reliability evaluation, and large subsets of our examples actually deal with reliability.

A probabilistic assessment of the safety of software is a formidable task for the assessor, for which no proven methodology is available. Accepted standards and guidelines only help in checking that recommended or prescribed practices were applied in the development, verification and validation of the software, not that the result of their application is indeed the required level of safety. Mathematically rigorous conclusions are only possible from very specific types of evidence, e.g., results of statistical testing, and are often insufficient for certifying the required levels of safety [Littlewood and Strigini 1993, Strigini 1994a].

The assessor is confronted with a wealth of evidence about the design methods used, the quality assurance organisation, and the results of testing, none of which is in itself 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 this evidence into a statement that the software is safe (or reliable) enough. We believe that this step of integrating the available 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 obviate its necessity (for a more complete argument, see [Littlewood and Strigini 1993]).

In this judgement, experts rely on their previous experience as well as on the evidence about the individual project they are assessing. The net of cause-and-effect chains, deductions and inferences which binds the evidence with the conclusion to be reached is presumably rational, but so complex as to defy analytical description. In trusting engineering judgement, we rely on the ability of the human mind's (and especially the experts' mind, through partially unconscious algorithms learnt from experience) in

[1] The term "safety case" is currently used in some fields of industry to indicate a *documented body of evidence organised to provide a convincing and valid argument that a system is adequately safe for a given application in a given environment (paraphrased from [Bishop, 1994]).*

deriving decisions from such complex mazes of evidence and reasoning. However, these abilities cannot be taken for granted: there is abundant literature (from experimental psychology) showing the fallibility of various categories of experts in such tasks [Strigini 1994b]. In addition we must consider that safety assessment often concerns rare events for which paste experience may simply be insufficient to draw any strong conclusion.

> The assessor thus has to rely on engineering judgement, but this judgement is an obscure, hidden psychological process and it should not be trusted lightly. To quote Richard Feynman, "As far as I can tell, 'engineering judgement' means they are just going to make up numbers" [Feynman 1988]. But engineering judgement does not need to be trusted blindly, if the actual reasoning of the expert, or the rigorous reasoning for which the expert's quick insight provides a shortcut, can be described, and thus checked and criticised by the same or by other experts [Strigini 1994b]. Unfortunately, this is usually not done because of the daunting complexity of the task.

To facilitate the task of the assessor, we have looked for ways of formalising the software safety case, i.e., giving an explicit formulation of the reasoning of the assessor, susceptible of a rigorous mathematical interpretation. The benefits we would expect from such "formalisation" are that the assessor can double-check his/her own reasoning for consistency and plausibility of its premises and deductions, and the safety case is open for inspection by other assessors, corporate decision makers, licensing authorities, etc. The consistency of the arguments can be checked by automatic tools, so that the human assessors can concentrate on the correctness of the premises and the structure of the reasoning, rather than the verification of the complex computations involved.

These methods, if supported by enough statistical evidence (e.g., about the effectiveness of software engineering methods) could raise the levels of safety of the software that can be reliably certified. In particular, they would allow the assessor to take into account all the evidence available, while preserving logical and mathematical rigour. Even when modest levels of safety are assessed, 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 experts' opinion, without the possibility of verifying the reasoning which led from the existing evidence to that opinion.

We chose the formalism that is variously called "belief networks", "probabilistic belief networks", "Bayesian causal networks", and such. This formalism, with software tools allowing its use by non-mathematicians, is being increasingly applied to decision problems in different fields [Hall *et al.* 1992, Kleiter 1992, Reed 1993]. It is based on the calculus of probabilities, which seems 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. This formalism is based on a Bayesian approach to probability, i.e., the probability of an event is seen as the "degree of belief" that one can rationally hold about the event taking place. The calculus of probabilities can then be used for deduction from premises that are statements about the probabilities of events rather than deterministic statements about events taking place or not taking place. In particular, statistical inference from observed facts is treated uniformly with the rest of probabilistic reasoning (of which deterministic arguments and proofs are a special case). Automated tools are available for manipulating belief networks (in general, these tools were originally intended for the construction of expert systems using Bayesian calculus to deal with uncertainty).

> This formalism and its supporting tools seems to have a potential for helping assessors with many of their current problems: integrating "process-related" and "product-related" reasoning; integrating structure-based probabilistic modelling with statistical inference; building rigorous arguments for critical "lemmas" in a safety case, while retaining the possibility of integrating these into larger arguments at a later stage if necessary; building safety cases that can be used throughout the lifetime of a product, updated

with new evidence, detailed where necessary by adding new sub-arguments or refining pre-existing ones; building re-usable templates of the safety cases, each appropriate for use in a specific category of situations (application, organisations, etc.), making these easy to use for (non-mathematician) domain experts but retaining the ability to access, check and modify the underlying mathematical structure.

To know whether belief networks are appropriate *in practice* as an aid in software safety cases, one needs to investigate whether their formalism is reasonably easy for specialists in the software dependability field, and whether it lends itself to representing the kind of information that are available in a software evaluation exercise.

This paper is meant to provide a first supporting argument for the appropriateness of Bayesian networks as an aid for the assessment of software dependability, and for the feasibility of their use. It is based on simple exercises in using probabilistic belief networks for reasoning about software dependability. These exercises use plausible scenarios and numerical values, but are not based on actual software projects, nor do they represent a complete safety case. Our purpose in performing these exercises is to evaluate whether the application of this method to an industrial case study would be worthwhile, and to obtain indications for guiding it. For instance, we are interested in learning which structures of belief networks would be useful and how the kinds of statistical knowledge that are typically available could be integrated into them.

In section 2, we recall the problems of software evaluation, and in section 3 we briefly introduce Bayesian reasoning and belief networks. Section 4 contains two simple examples; Section 5 discusses the value of belief networks or similar formal mathematical reasoning for a safety case. Section 6 develops in some more detail an example belief network, showing some of the issues made explicit in defining the belief network, the difficulties of the task and the results to be expected. Section 7 presents our conclusions.

2 Problems in probabilistic assessment of software

The difficulties of obtaining trustworthy predictions of software dependability are well known. Claims of high dependability are usually based on a combination of diverse evidence, like test results, use of good (i.e., conducive to high reliability) programming languages and tools, existence of a quality assurance or safety plan, etc. Of these elements, only test results are direct evidence about the dependability of the product. The other elements (mostly concerning the software production *process*) may indicate that the dependability measures for the product are likely to be in a certain range; but they certainly do not allow very precise predictions. It has been observed (see e.g. [Fenton 1993, Fenton *et al.* 1994] that most claims about the usefulness of software engineering methods and tools are supported by very little statistical evidence. Even such popular principles like structured programming have been accepted on the basis of convincing advocacy rather than good experimental science. On the other hand, whatever little support (for predicting high levels of dependability) can be obtained from process-related evidence is needed, as the dependability requirements for critical software products often exceed what can be demonstrated by testing only.

The arguments that can be built with process-related evidence are usually of a complex probabilistic nature: for instance, a good programming language does not absolutely prevent programming errors. What it does is enhancing the expectation that in most projects, if coupled with other favourable conditions (e.g., good management and project scheduling, competent programmers) it will produce few (possibly zero) faults, and programs will be unlikely to contain many faults, unless the choice of language is accompanied by seriously detrimental conditions (like excessive time pressure and inexpert personnel). This expectation that using a certain programming language will lead to generally less error-prone (than is otherwise common) programming may be justified, in its turn, by the fact be that this programming language has been observed to lead to fewer errors. The latter is a statistical argument, which is more or less strong - and thus raises our expectation by different amounts- depending on the size and representativity of the sample observed. Another supporting argument may be based on a model of the mental activities involved in programming, the patterns of errors and the way these are affected by the features of programming languages; in this case, statistical supporting arguments are needed separately for the different parts of the model. Other evidence in a safety case will be of a deterministic nature, e.g. a proof that the product is incapable of certain kinds of erroneous behaviour, provided that some hypotheses hold. The statement that the hypotheses hold may in turn be supported by a probabilistic argument (see examples in [Powell 1992]), and so on. These arguments are thus by necessity involved and difficult to verify, which explains the frequent recourse to "engineering judgement" as a label for incompletely developed arguments.

We believe that a way forward lies in stating the arguments with enough precision to make them amenable to thorough verification and criticism. Clearly, the formalisation process cannot be based on a simple recipe, but must depend on the details of the evidence available. For instance, the implication of the knowledge that a certain set of software engineering methods and tools have been used may include, for different tools, such diverse consequences as, e.g.:

- software bugs of a certain category are guaranteed to be absent (with a high probability, given by the confidence that a certain tool deterministically avoids or eliminates such bugs);
- some software bugs are likely to be rarer than if a certain tool had not been used, as shown by experience on other projects (and the significance of this experience for drawing conclusions about the current project is itself to be deduced through complex reasoning);
- a certain method is thought to improve reliability, but this belief is based on conjectures about the behaviour of the programmers that have never been tested experimentally; etc.

The supporting general knowledge that can be used to step from the raw evidence to dependability claims may range from physical laws, to general deterministic proofs of properties of the software, to probabilistic laws derived either from proof (e.g. about the effectiveness of a cryptographic algorithm or an error-detecting code) or from statistical evidence (e.g., about the effectiveness of a certain debugging technique). Managing the complex logical structures that result from these interrelated arguments (keeping track of which factors affect which other factors) is a difficult task in itself (see for instance [Kailar *et al.* 1994] for an attempt to trace all the sub-arguments needed for the rigorous assessment of cryptographic protocols). After tracking the logical dependencies, some calculus is needed to deduce whichever predictions can be deduced from all the raw evidence available.

Different formalisms (each with a calculus of its own) have been developed for reasoning with uncertainty [Ng and Abramson 1990, Saffiotti *et al.* 1992, Wright and Cai 1994], the most popular ones being Bayesian calculus [Cheeseman 1988, Pearl 1988], fuzzy theory [Zadeh 1975] and possibility theory [Zadeh 1978, Dubois and Prade 1988]. We explore the use of Bayesian calculus, for which there are both a rigorous mathematical definition and plausible claims that it offers support for optimal decisions. Bayesian calculus can be criticised in that it requires the available knowledge to be expressed in forms that may be alien to the intended users of the method, and such that the translation may be prohibitively difficult. Whether these problems are serious in our particular context will have to be investigated by experiment.

Reasoning about the effects of process factors on software safety may imply reasoning about their effect on static characteristics of the product (e.g. design faults), and of the latter on the behaviour of the product (as a function of the particular usage profile to which the product will be subjected). Reasoning about safety may imply reasoning in terms of reliability of the product, and of the more or less dangerous effects of failures, as a function of the usage profile and the characteristics of the usage environment.

A formally structured argument is not in itself a guarantee of successful evaluation, if by this we mean obtaining a prediction of satisfactory dependability. The formalism and calculus only facilitate the use of existing knowledge, and thus make it easier to avoid erroneous deductions. They cannot compensate for missing knowledge: no sophisticated mathematical treatment can prove, say, that a product which has not been extensively tested, and was produced by some new, untested method, is very dependable. It is also important to choose which knowledge will be included in the argument: one could try to model all kinds of subtle relationships between characteristics of the product and process, but many of these will not increase the levels of dependability that can be claimed, while they will multiply the effort of collecting and describing the evidence. While checking the consistency and correctness of arguments can be left to an automated assistant, choosing which evidence, and which knowledge about the implications of such evidence, is to be included in the argument requires competent human decisions. In this, again, the tools for formalised description of reasoning can be useful, because the user can tentatively add more detailed knowledge to check its possible implications, and thus better decide which level is convenient in the formal representation of the argument.

3 Belief networks

In recent years, much attention has been directed at probabilistic reasoning in graphical models. Graphical models are known under various synonyms as : Belief Networks, Causal Probabilistic Networks, Bayesian Belief Networks, Causal Nets, Probabilistic Cause-Effect Models, Probabilistic Influence Diagrams etc. They have been used in a wide variety of applications [Pearl 1993, CACM 1995] as an appropriate representation for probabilistic knowledge. Researchers from the communities of AI, Statistics, Medicine, Decision Analysis are interested in the issue of probabilistic reasoning. In addition to theoretical investigation [Pearl 1986, Lauritzen and Spiegelhalter 1988], software products have been built (HUGIN [Andersen *et al.* 1989], ERGO [Herskovits 1994], DEMOS [Henrion *et al.* 1991]) to support the use of these notations. These products were used early on in medical environments, with application later extending to various fields [Reed 1993]. What follows is a very brief introduction to the topic. The interested reader can turn to ([Charniak 1991, Henrion *et al.* 1991, Jensen 1993, CACM 1995, Wright 1995]) for more complete information.

A belief network is a directed acyclic graph used to represent probabilistic relationships among events. A key characteristics of this formalism is its ability to deal with the *inherent uncertainty* of our knowledge about the real world. We represent our probabilistic knowledge about a certain real-world situation by specifying: i) the topology of the belief network and ii) probability tables associated with its nodes. This model of our knowledge also represents the arguments that can be built to support a thesis about the probabilities of some events in the belief network, 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 supports 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.

In a belief network, each node represents a set of events (a partition on the set of outcomes of an experiment or observation), possibly a numerical random variable. The events, or values of the random variables, 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. With each node, a table of probabilities is associated. If the node has no incoming arcs (root node), this table lists the ["marginal"] probabilities of the possible values of the node; if it has incoming arcs, the table is a table of conditional probabilities (of the values of this node, conditional on the values of its "parent" nodes). Arcs and tables of conditional probabilities can thus be used to represent the fact that knowledge about one node is useful for predictions about another node: through cause-effect relationships ("amount of debugging" and "number of bugs before debugging" affect "bugs left after debugging"), or via more general correlation laws ("which team developed this software" affects "number of bugs"). Once one has derived 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, from the tables of conditional and marginal probabilities of the ancestor nodes, the probabilities of events represented by nodes with incoming arcs;
- 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).

A typical use of arcs is to indicate causal links between the factors represented by nodes: this intuitive meaning is what gives belief network (often called "causal networks") their popularity in the AI field. However, the arcs may represent any kind of probabilistic correlation: causal, diagnostic (from effect to cause, rather than from cause to effect), or simply an observed statistical correlation that is believed always to hold. An arc between nodes A and B (in either direction) indicates simply that if we know the state of A our probabilistic knowledge about B (our subjective probabilities for the states of B) is different than it would otherwise be. This information can be described in mathematical terms by giving the probabilities of the states of A conditional on those of B, or vice-versa, or via correlation coefficients between states of A and B: all these representations are equivalent. People reasoning in intuitive terms often treat diagnostic correlation very differently from casual correlation (they treat diagnostic evidence as weaker than causal evidence). It is thus useful that the language of belief networks allows them to represent those correlation links that they best understand, and let the support tools translate them into the single underlying mathematical representation, using, for instance, causal information for diagnostic inference as required.

Choosing the conditional and marginal probabilities to be associated with the nodes is obviously difficult, and this difficulty is probably the main perceived obstacle to a wider application of Belief Nets. 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. For the former case, tool developers are now enriching their tools with the ability to automatically derive probabilistic knowledge from relevant databases [Cooper and Herskovits 1991, Lam and Bacchus 1994, Olesen and al 1994], so that the necessary knowledge can be automatically derived from large collections of data about past cases. By comparison, the choice of the nodes and the arcs to be included in a belief network appears to be relatively easy. However, the precise mathematical implications of choosing a certain network topology may be hard to grasp intuitively. The arcs entering a node imply a conditioning of the probabilities of the states of that node only on the states of its "parent" nodes: the states of the parent nodes are enough to determine the probabilities of the states of the child node. So, the absence of arcs between two nodes is a statement of conditional independence between those two nodes (conditional on the states of some other nodes).

In other words, a network topology implies that the joint probability distribution of all the variables in the belief network can be decomposed in terms of the set of marginal and conditional distributions associated with that topology. Any topology with additional arcs (and many with different sets of arcs) could still (with appropriate probability tables) represent the same joint distribution, but would fail to convey the visual information that some nodes provide no additional knowledge for statements about other nodes.

To reason in probabilistic terms we need to identify events of interest and random variables related to events. The random variables must be chosen so as to be useful in reasoning. For instance, a variable which cannot be measured, and about which we do not know how it affects, or is affected by, other variables, is normally useless. So, the task itself of selecting the nodes and arcs in a belief network will help one to focus on the precise meanings of the concepts involved, the nature of the relationships (causal, diagnostic, or other) between different concepts, and the availability or lack of factual knowledge concerning these relationships.

4 Introductory examples

We give in this section two simple examples of Bayesian inference and of the use of belief networks. These examples are simple enough that one would not need Bayesian networks to follow their statements and solutions: hence their use as introductory examples.

4.1 A textbook example: diagnosis

A textbook example of the use of Bayes' theorem is in modelling diagnosis: e.g., what is the probability that a patient who tests positive for a disease actually has the disease? For a software-related parallel, we consider the following example.

We are developing a system, and know that there is certain bug type appearing in 3% of the modules (this is our prior knowledge). The problem is deciding whether a certain specific module contains the bug. We have a diagnostic test (for instance, a code inspection procedure, an automated dynamic test procedure) for detecting that kind of bug. From past experience, we know that, if a software module contains the bug, our test will reveal an anomaly with a probability of 0.87. However, in 1 % of the modules that do *not* contain the bug, the test will detect an anomaly as though the bug were present. This test procedure is still valuable, as it is a straightforward way of signalling a possible source of serious problems. However, if it detects an anomaly, what is the probability that the bug really exists? We can represent our knowledge so far in the following table of *conditional probabilities*.

Given this true state of the module:	The test outcome will be	
	Anomaly detected	No anomaly detected
	.... with these (conditional) probabilities:	
Bug Absent	0.01	0.99
Bug Present	0.87	0.13

Table 4.1 Conditional probabilities of test outcomes, conditional on the bug being and not being present

To answer our question, we need the additional piece of information stated before, i.e., the probability that the bug exists in a module chosen at random (without any additional knowledge to differentiate this individual module from the rest of the population) is 3 %. In the terminology commonly used in Bayesian analyses, this is our *prior probability* that the module contains the bug. Having stated all these probabilities, we have enough data for filling in the table below.

		True State of the Software Module		Total per row
		Bug Absent	Bug Present	(probabilities of test outcomes):
Outcome of test	Anomaly detected	0.97*0.01= 0.97 %	0.03*0.87=2.61 %	3.58 %
	No anomaly detected	0.97*0.99=96.03 %	0.03*0.13=0.39 %	96.42 %
Total per column (probabilities of true states of module)		97 %	3 %	100 %

Table 4.2 Probabilities of the 4 possible situations, given the conditional probabilities shown above and a 3 % prevalence of the bug.

If, at this point, our test detects an anomaly in our module, the probability that the module really contains the bug is given by the ratio of the probabilities of the two events ("the test detects an anomaly, and the bug is indeed there" (0.0261) and "the test detects an anomaly" (0.0358), i.e., about 0.73. This probability of the bug being present, conditional on the test detecting an anomaly, is called the *posterior* probability that the software contains the bug ("posterior", i.e., successive, to obtaining additional information via the test). For each outcome of the test, this procedure (which is an application of Bayes' theorem) yields the posterior probability that the bug is or is not present, shown in the table below.

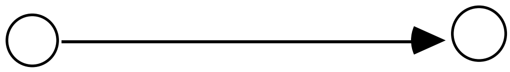
	The probability that the bug is really	
	present	absent
If the test says:	is:	
Anomaly detected	~ 72.9 %	~ 27.1 %
No anomaly detected	~ 0.4 %	~ 99.6 %

Table 4.3 Posterior probabilities of bug being present, given test result

Clearly, for the probabilistic model to yield correct predictions, the proper choice of prior probabilities is just as important as that of the conditional probabilities.

> The fact that the test, when it says that the bug is present, is wrong 27 % of the time is at times seen as counter-intuitive. It is indeed common for people to assume at first that the test can only be wrong 13 % of the time, confusing between the conditional and the marginal probabilities. One can also notice that when the test results is the "usual" one ("Bug not detected"), the resulting posterior probabilities are not very different from the prior probabilities. It is the relatively rare observation, i.e. the test output "Bug detected", that dramatically changes our beliefs.

This simple reasoning structure would be represented by the following belief net, with two nodes (each with two possible states) and one arc:



True state of the module
("Bug is present"
or
"Bug is not present")

Test results
("Anomaly detected"
or
"No anomaly detected")

Fig. 4.1

To the node on the left, one would associate the marginal probabilities of its two possible states, and to the node on the right, the conditional probabilities from our first table. The computer tools available for manipulating belief networks would then be able to calculate any consequence of this initial setting and of any additional observation about the facts represented. For this network, they automatically compute our prior distributions of the states of the nodes on the right (marginal probabilities of test outcomes - totals for the rows of our second table); they allow the user to state e.g. that the state of the right-hand node is "Anomaly detected", and the tool then automatically computes the posterior probabilities for the node on the left, as in the right-hand column of our third table.

4.2 Statistical testing example

A software product is developed with a certain requirement on its probability of failure on demand (Pfd). Then, to test that it satisfies this requirement, the software is subjected to statistical testing (with inputs chosen as independent samples from an input distribution representative of the intended operational use of the software). If the testing reveals any fault, the software will be subject to further debugging. If the testing reveals no fault, however, we are interested in knowing what probability of failure on demand we should expect from the software.

In this case, we represent our knowledge in the following 4-node network. We have assumed here that the Pfd may take on a discrete series of values: 0 (perfect software), 10^{-6}, 10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}. This is clearly an approximation of a "reasonable belief", which would clearly take the form of a continuous distribution (there is no reason why the true Pfd should not be, for instance, 0.008). Requiring that the event described in each node of a belief network may assume only a finite number of values is an unfortunate artefact of the computer tools we are using, which may be alleviated by subdividing the real axis in as many intervals as is convenient. On the other hand, an approximation with only few possible values may often be more natural for the expert who has to express his/her prior beliefs, and be sufficient for an initial check of the probabilistic argument. We have only used six values to keep the example reasonably simple. Notice that considering a Pfd>10^{-2} completely impossible is a strong assumption. According to the rules of Bayesian calculus, it will prevent us from accepting even a small probability of such values, even if confronted with much evidence in that sense. Such assumptions should be made cautiously, only on the basis of strong arguments, or, even better, one should check that their effect on the specific conclusions one reached through the use of the belief network is negligible. The belief network we choose is as follows:

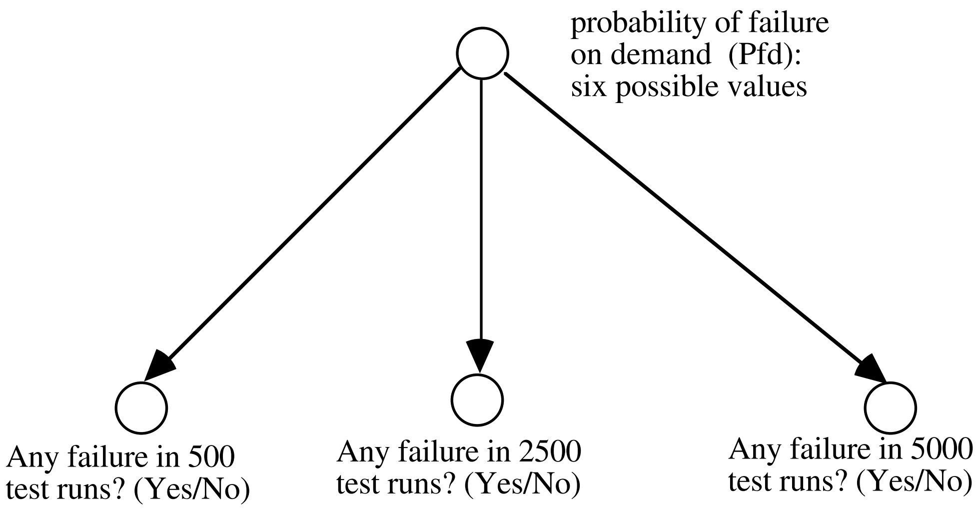


Fig. 4.2.

For a given value of the Pfd, we derive the conditional probabilities of the two outcomes at each of the lower three nodes using the formula (justified by the fact that tests are independent trials - Bernoulli trials):

P(no failures in a run of N tests | Pfd=p) = $(1\text{-p})^N$

The choice of prior probabilities or probability distributions is the most sensitive part of the exercise. Whatever assignment we choose, it must be supported by some rational argument. We first explore the case in which the six possible values of the Pfd are considered (prior to testing) equally likely (each has probability 1/6).

> This could be described as a description of "ignorance"; but this is just an artefact of our choice of letting the Pfd take only these six possible values. In general, there is no unique "ignorance" prior. In this case, plausible alternatives would be, for instance: equal probabilities for each among many Pfd values in the sequence 0.1, 0.01, .. ; a uniform probability density function along the interval [0,1].

	Conditional probabilities of values of the leaf nodes					
	After 500 tests		After 2500 tests		After 5000 tests	
Pfd Value	0 failures	≥ 1 failure	0 failures	≥1 failure	0 failures	≥1 failure
0	1	0	1	0	1	0
0.000001	~1	5.00E-04	9.98E-01	2.50E-03	9.95E-01	4.99E-03
0.00001	9.95E-01	4.99E-03	9.75E-01	2.47E-02	9.51E-01	4.88E-02
0.0001	9.51E-01	4.88E-02	7.79E-01	2.21E-01	6.07E-01	3.93E-01
0.001	6.06E-01	3.94E-01	8.20E-02	9.18E-01	6.72E-03	9.93E-01
0.01	6.57E-03	9.93E-01	1.22E-11	~1	1.50E-22	~1

Table 4.4: Conditional Probabilities of Leaf Nodes, given the 6 possible values for the Pfd.

This prior belief for the top node also determines the prior probabilities for the other events (through their conditional probabilities - conditional on the value of the top node - which we already assigned). We can now proceed to consider how different test outcomes would change our prior beliefs. Figure 4.3 depicts the initial allocation of belief in our model, with equal priors for the six values of the Pfd. The following figures represent the effects of observing the outcome of one of the three experiments represented by the three bottom nodes.

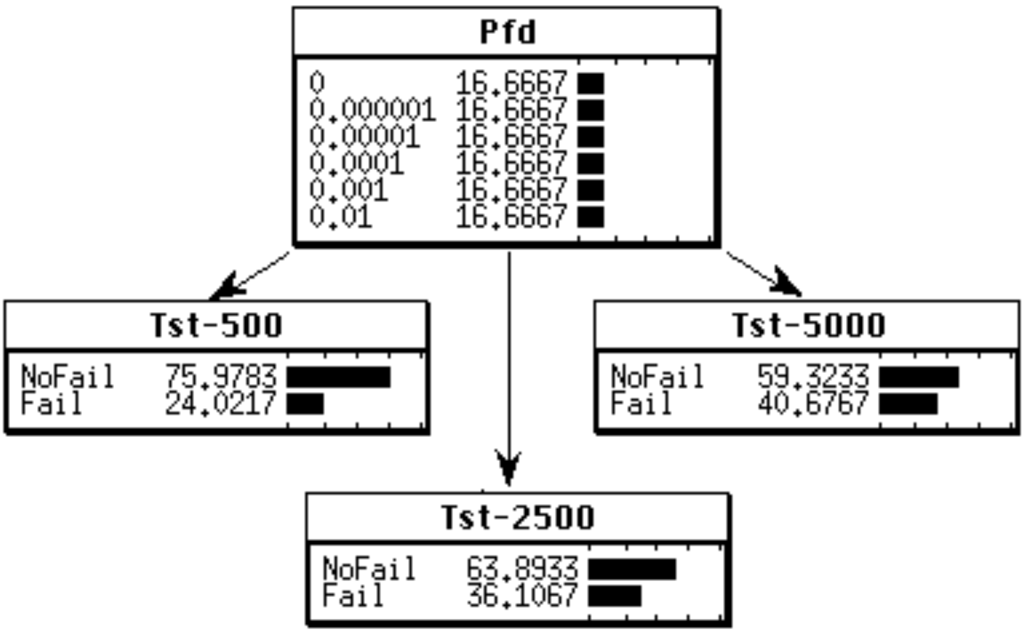


Fig. 4.3 Prior probabilities for the network in Figure 4.2, if the six values of the probability of failure on demand have equal prior marginal probabilities.

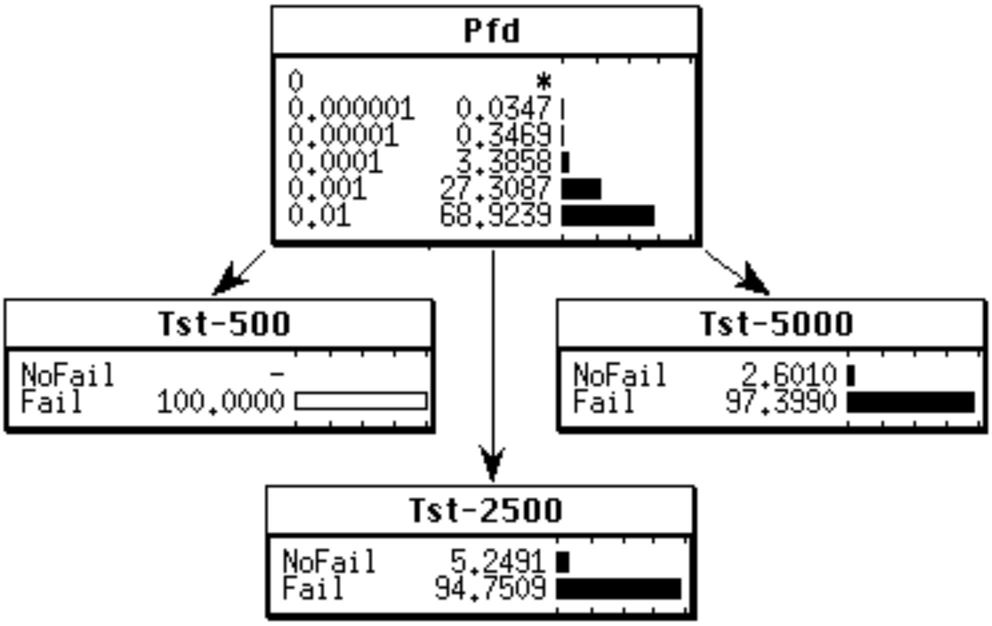


Figure 4.4 : Failure Encountered in 500 tests. Our posterior beliefs are drastically different from our priors: it is very likely that the Pfd is as bad as 0.01, and the predictions for the outcomes of 2500 and 5000 tests are changed to near-certainty of failure.

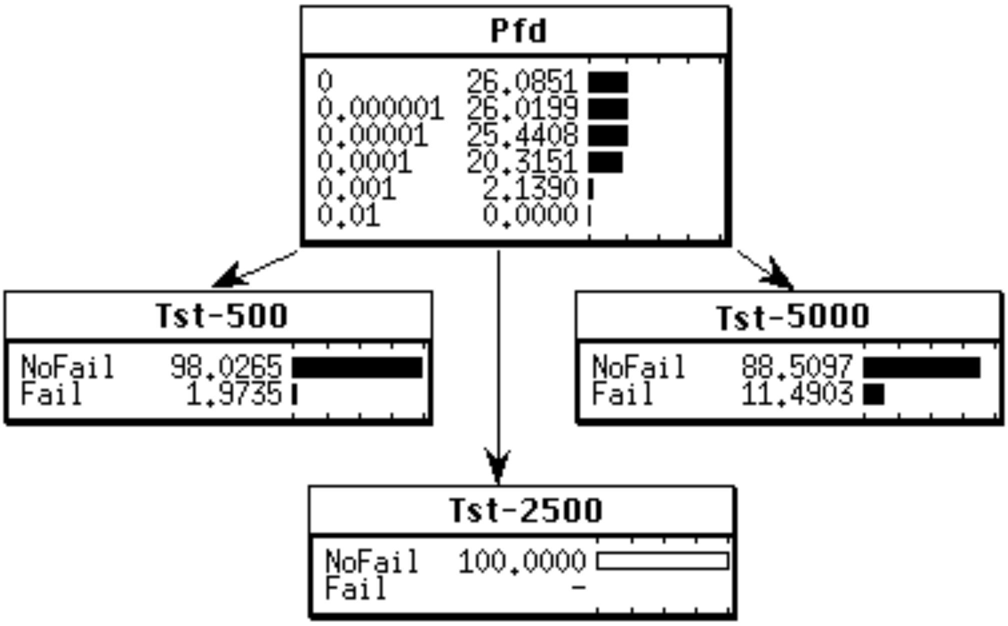


Figure 4.5 : No Failure in 2500 tests. The posterior distribution favours very low values of Pfd.

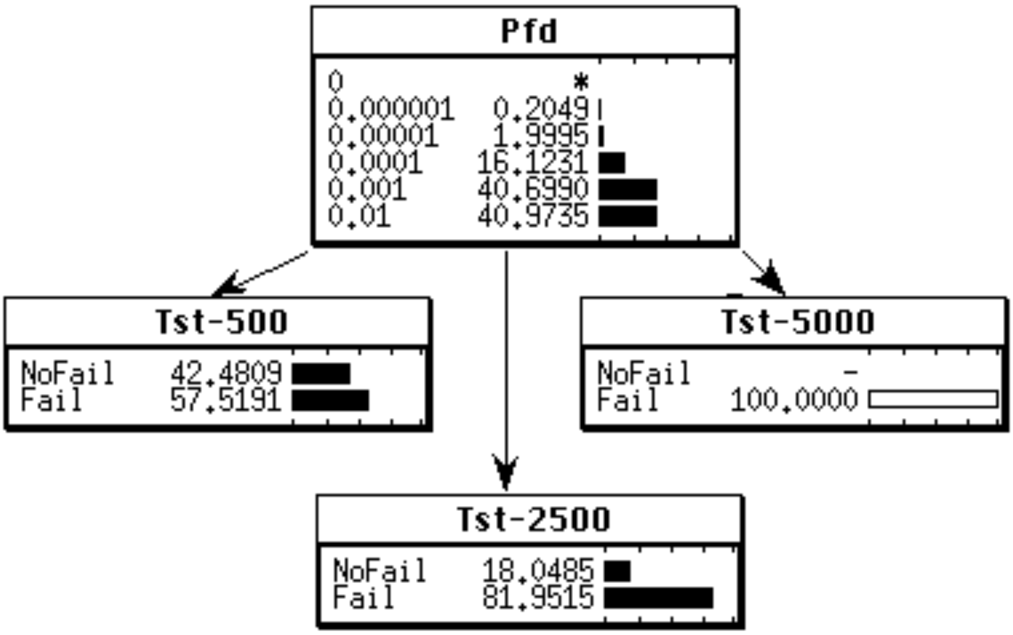


Figure 4.6 : Failure in 5000 tests. This removes the possibility of perfect software, while giving nearly equal chances to the two non-zero values of Pfd=10^{-2} and Pfd=10^{-3}. This evidence implies that the shorter test runs would be likely to produce a failure as well. However, if we intend to draw precise inference from observing failures, we could obtain more precise indications by discriminating, in the belief network, between the cases in which 1, 2, 3, ... failures are observed.

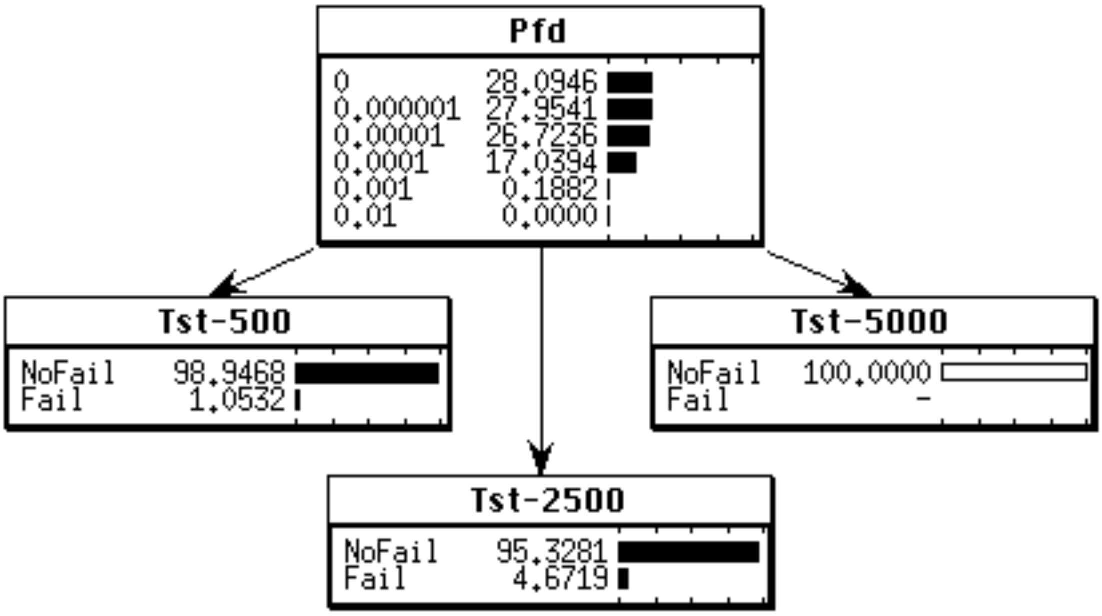


Figure 4.7 : No failure in 5000 tests. The model gives the majority of probability mass to the conjecture of fault-free software. One should suspect this to be an artefact of our choosing 10^{-6} as the lowest possible Pfd for non-perfect software. However, the Pfd is almost certainly lower than 10-3.

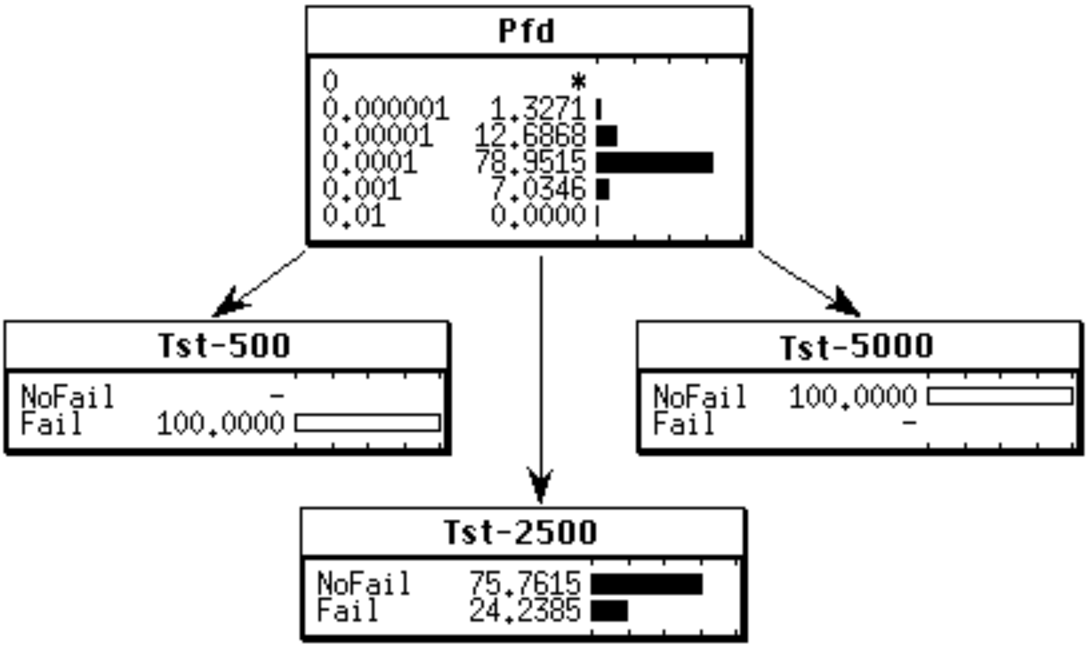


Figure 4.8 : Two independent test series were performed, one with 500 tests (failure observed) and the other with 5000 (no failures). The overall result is that the software cannot be fault-free, yet the belief that it has a low failure rate is reinforced.

As we pointed out before, the choice and representation of prior beliefs affects the posterior beliefs derived from observation of new evidence. We underscore that this is not a weakness of Bayesian methods, but a strength: "classical" statistical inference, yielding "confidence levels" for conjectures, deals with the probability of observing the evidence if the conjecture were true, but not with the probability that the conjecture is true after the evidence is observed. The latter can only be determined as a function of a prior probability. It is the evaluator's responsibility to use prior beliefs that are based on sound scientific evidence, or to show that varying the prior beliefs within a plausible range does not alter the posterior beliefs so as to invalidate any conclusion of interest.

If we have to base our priors on the opinions of experts, without the benefit of more scientific support, the Bayesian network can be used by the experts to check the consistency of their beliefs. For instance, the table below would tell the experts how likely they "should" consider a certain test outcome to be (shown in one column in the bottom section of the table), if their prior beliefs about Pfd are really as represented in the same column in the top part of the table. This consistency check works both ways, of course: one could try and elicit an expert's belief about the distribution of Pfd by asking questions about the probabilities of the outcomes of different numbers of tests.

Pfd Value	Prior I	Prior II	Prior III	Prior IV
0	16.67	1	70	30
0.000001	16.67	10	10	25
0.00001	16.67	20	5	20
0.0001	16.67	40	5	15
0.001	16.67	19	5	9
0.01	16.67	10	5	1
Probabilities of Test Outcomes:				
500 Tests, No Failure	75.98	80.53	92.79	94.62
500 Tests, Failure	24.02	19.47	7.21	5.38
2500 Tests, No Failure	63.89	63.19	89.16	86.86
2500 Tests, Failure	36.11	36.81	10.84	13.14
5000 Tests, No Failure	59.32	54.36	87.77	83.06
5000 Tests, Failure	40.68	45.64	12.23	16.94

Table 4.5: Various prior distributions for the Pfd, compared on the basis of beliefs about test outcomes.

5 Uses of formalised probabilistic safety arguments: proof vs. rigorous critique

It seems that the "formalisation" of safety arguments (e.g. via belief networks) can be put to two main uses, which may coexist in any specific application

5.1 Formal arguments as proofs

We list this use first, as it is the more obvious one: one uses the formalisation exercise actually to prove the safety claims made. The belief network must then completely describe all the relevant parts of the argument linking the evidence available to the conclusion of the argument, i.e., the safety claim.

In many cases (at least when the claim takes the form of a very low probability of dangerous failure), this task would be very difficult, because of the complexity of the network, the need to fill in conditional probabilities describing arcane relationships between the relevant variables (e.g., how, exactly, does the experience of the development team affect the proportion of failures which produce a certain unsafe behaviour?), and, last but not least, the insufficiency of the statistical data available (about test results, previous experience with similar method and products, etc.).

In other cases (all those in which the available evidence does warrant belief in the claimed level of safety, we would hope), the belief network will provide a complete proof of the claim. In so doing, it will also provide much useful additional information, like for instance: which parts of the evidence used are essential to support the claim , and which are marginal or inconclusive. It will also provide a framework for updating the safety case with the accumulation of knowledge through the project and the lifetime of the product

> An aspect of dependability predictions is that the amount of knowledge on which they can be based increases during the life cycle of a product. Thus, an initial feasibility study can only infer the dependability to be expected on the basis of past experience with similar products (or observed effects of the different factors involved); as development proceeds, new data (e.g. fault counts in different stages of verification, complexity-related measurements) will give cues as to the position of the actual project in the spectrum of scenarios contemplated in the feasibility study; the reliability growth during debugging, the results of operational testing and, last, the accumulation of operational experience will give additional, stronger indications of the dependability to be expected in subsequent use. Thus, a safety case should be an evolving document, revised periodically to take account of all lessons learned in the intervening time. An

interesting feature of Bayesian reasoning is that the distributions of all variables of interest, initially assigned on the basis of whichever knowledge is available, can be refined with observation. All kinds of variables that one chooses to use in the reasoning - e.g., the number of faults discovered in a certain phase, the number of failures observed in a given period of use, the amount of code in the product, etc. - can be assigned distributions from the very beginning. Many of these distributions will not be sufficiently informative to be useful. As the project proceeds, the values of an increasing number of these variables will be observed, causing (through the Bayesian inference procedure) all other distributions to be updated.

5.2 Formal arguments as a basis for criticism and insight

Another use is possible for the formalisation exercise, however. If it is not feasible to build such a complete rigorous case, and the assessment is thus based on the experts' informal reasoning, building "hypothetical" arguments can be a powerful aid for validating such informal reasoning. For instance, a "hypothetical" belief network may link only a few of the relevant variables, and some of the probability tables may be filled, in the absence of sufficient statistical knowledge, with "hypothetical" probability distributions chosen in order to produce the same conclusion reached by the expert assessor. The assessor can then check whether the structure and the probability distributions in this hypothetical argument appear to be plausible and/or represent his/her own reasoning. This exercise amounts to making sure that the expert's belief about the final claim of interest (say, that a certain undesired event is very unlikely) and his/her "supporting" beliefs (marginal probabilities for the root nodes, and conditional probabilities for all others) are consistent: whether, for instance, the claim is actually a logical consequence of the supporting beliefs. The automated tools will guarantee the mathematical consistency of the arguments, so that the problem of deriving the consequences of the stated distributions, and checking such complex structures for inconsistencies is largely removed. The experts can thus look for a belief network that they believe to be sound and that supports their own conclusions. If no such argument can be found, the conclusions may be wrong. If it is found, and it depends on some probability distribution that is unsupported by statistical data, but believed plausible, then data collection can be focused to obtain the necessary statistical evidence.

This use of formal reasoning is just as important as the previous one. It would be a mistake to believe that only complete proofs make full use of the potential of mathematics. Quite the opposite is true: in the physical sciences, no belief can be based on mathematical proof alone, without empirical validation of the premises on which the proof is based; in addition, empirical validation amounts only to continually challenging and weeding out incorrect conjectures, rather than supporting an absolute belief in any one of them. The function of mathematics is that they allow us to organise our understanding of the physical world and of the relationships between our conjectured theories and the observed or observable facts. In the case of safety evaluation, our conjecture is that the system is safe enough, and validating this conjecture means challenging it, by checking that its implications do not contradict empirical knowledge. In the end, the value of proofs is that building them requires us to understand the problem, not that they definitely prove any conclusion about real-world events. The "complete proofs" described in the previous section may only be called "complete" insofar as their premises are consensually accepted. The difference between that situation and that outlined in this section is one of degree (of the uncertainty of the premises), not of kind.

In our exercise, we have actually noticed many possible uses of the belief network tools, e.g.:

- communicating easily (visually) models of probabilistic dependencies and conditional independence in complex probabilistic models;
- deriving consequences of specific probabilistic beliefs (data generation), to be exposed to corroboration or refutation by experiment or checks of plausibility;

- deriving alternative statements of complex joint probability distributions from their statement in the form of a given network topology and its associated probability distributions;
- refining prior probabilities using the data collected on an individual project ("specialising" one's generic knowledge for an individual case);
- inferring the distributions of non-measurable attributes with a clear intuitive meaning, thus widening the range of conjectures that one could venture as a result of the observed data, for later empirical validation.

Some of these advantages are characteristic of any form of formalised reasoning, others of the probabilistic or of the Bayesian approach; others derive from the user-friendliness of visual representations of complex abstract systems, and of software tools that eliminate the burden of performing complex computations.

6. A more complete example

6.1 Modelled situation; the model

We now describe by a belief network a plausible structure for a software safety case. We reiterate that no universally appropriate structure exists for such a belief network. Our example is plausible, in that it refers to the information that could realistically be available in a software development organisation. However, this is a fictional scenario and organisation, built, like a novel character, out of realistic fragments. No attempt should be made to recognise in our fictional organisation any individual real-world organisation.

As we said, a sound and useful argument can only be built using the knowledge available in the interested organisation. So, we are not trying to show a "typical" or "general" example of formalised safety case, but, rather, a realistic example of the kind of reasoning which would be needed to build one in a situation (i.e., with a body of available knowledge) that is itself plausible, but just one of the many possible situations. For this study, we obtained the probability distributions that we used from published statistical data (from an academic experiment), as a way of completing a realistic exercise. We believe that no claim of "typicality" can be made for this specific set of numerical values, nor, actually, for any other set, in the present state of insufficient statistical knowledge about the software industry.

In our example, a safety case is built (as is common) on the two supporting arguments of excellence in development ("process" argument) and of failure-free statistical testing ("product" argument). The product (and process) under evaluation are considered to be fairly typical of a class of development projects with which the organisation has extensive previous experience. The safety claim is expressed in terms of a probability of failure per demand (Pfd).

Reasoning about a probability of failure on demand makes our example mathematically simpler than reasoning about, say continuous-time reliability. At the same time, this is a scenario of practical interest: our system could be an on-demand protection system, say a safety shut-down system (rather than, e.g., a continuous control system with safety requirements. In this context, one can in certain cases assume independence between successive demands, an assumption that greatly simplifies inference from test results. This applies at least when "one demand" means "the inputs cross the threshold into the alarm region, after which point the system has to perform a rather complex, long shutdown sequence. After this, starting up practically erases from the system all memory of previous events".

The development process, from initial requirements to delivery of the product, is represented in a stylised way, as in the figure below:

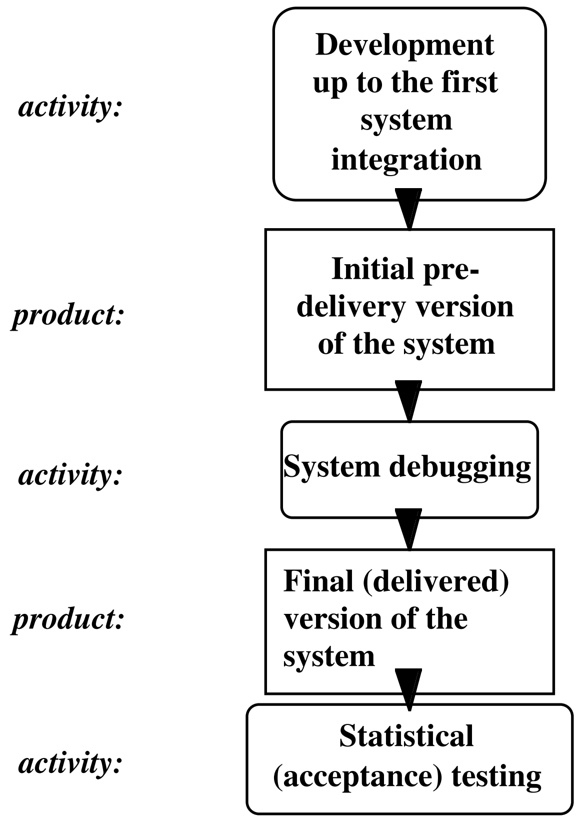


Fig. 6.1. Life-cycle phases of our hypothetical product.

The basic structure of the safety case is then as sketched in Fig. 6.2. Our case is 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 testing (showing no failures in a fixed number of runs, mandated by a contract or regulation) then corroborates this belief.

The detailed belief net is shown in Fig. 6.3. The process-related evidence (e.g., the skill of the development staff) is not explicitly represented by nodes and arcs in the belief network. It is represented, though, by assigning the prior probabilities for those variables that are represented explicitly on the basis of the distribution observed, for these variables, in a population of projects sharing the same process. 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 whole software system. 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.

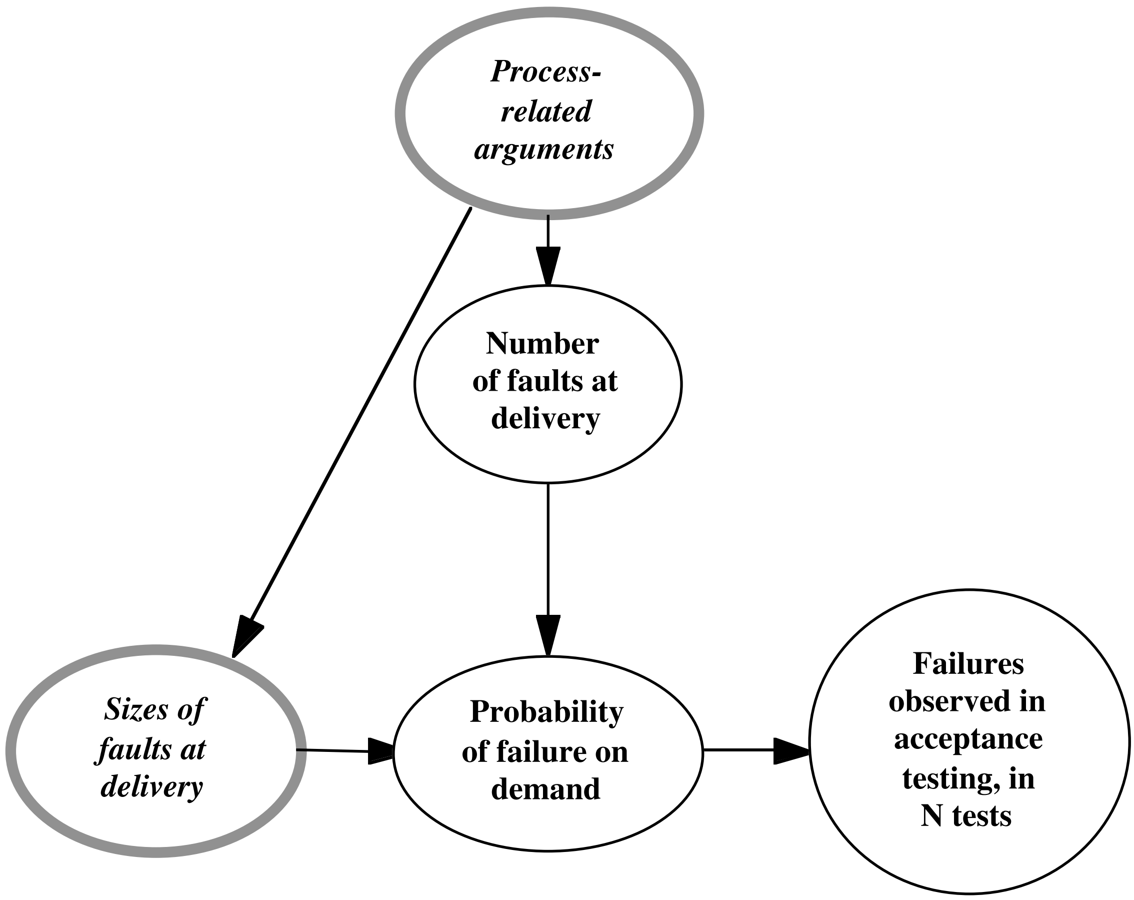


Fig. 6.2. A sketch of our safety case. The shaded nodes indicate intuitive concepts that need to be specified rigorously in order to build a belief network

We have thus built a comparatively simple belief net. We will now discuss it in more detail, showing that such a simple structure is reasonable for an organisation with comparatively little knowledge, and yet that the choice of the structure implies formal statements about the knowledge available and the causal relationships among the different factors.

At the top, the node "initial number of faults" (in the completed product before debugging) has a prior distribution based on measurements on previous products (of the same type and approximate size and complexity, and developed within the same organisation). Of course, these measurements must all be of an "initial number of faults" interpreted in a similar manner for all the data points. For instance, the data points could represent products that have been in use long enough that all the significant defects are thought to have been discovered.

The variables "Number of faults found during debugging" and "Number of faults at delivery" are of course related, as their sum must equal "Initial number of faults". We have chosen to use the corresponding nodes in the belief network as follows: the conditional probability distribution of "Number of faults found during debugging" given "Initial number of faults" is derived from the historical records of the organisation: this conditional distribution represents the observed effectiveness of the debugging process in this category of projects. The variable "Number of faults at delivery" is then a "deterministic" variable (in terms of the belief network, this means: for each given pair of states of its two parent nodes, the state of this node is known with certainty): adding this node to the belief network does not specify any new information about the distributions of the other variables, but is useful in that it represents a random variable with clear intuitive value.

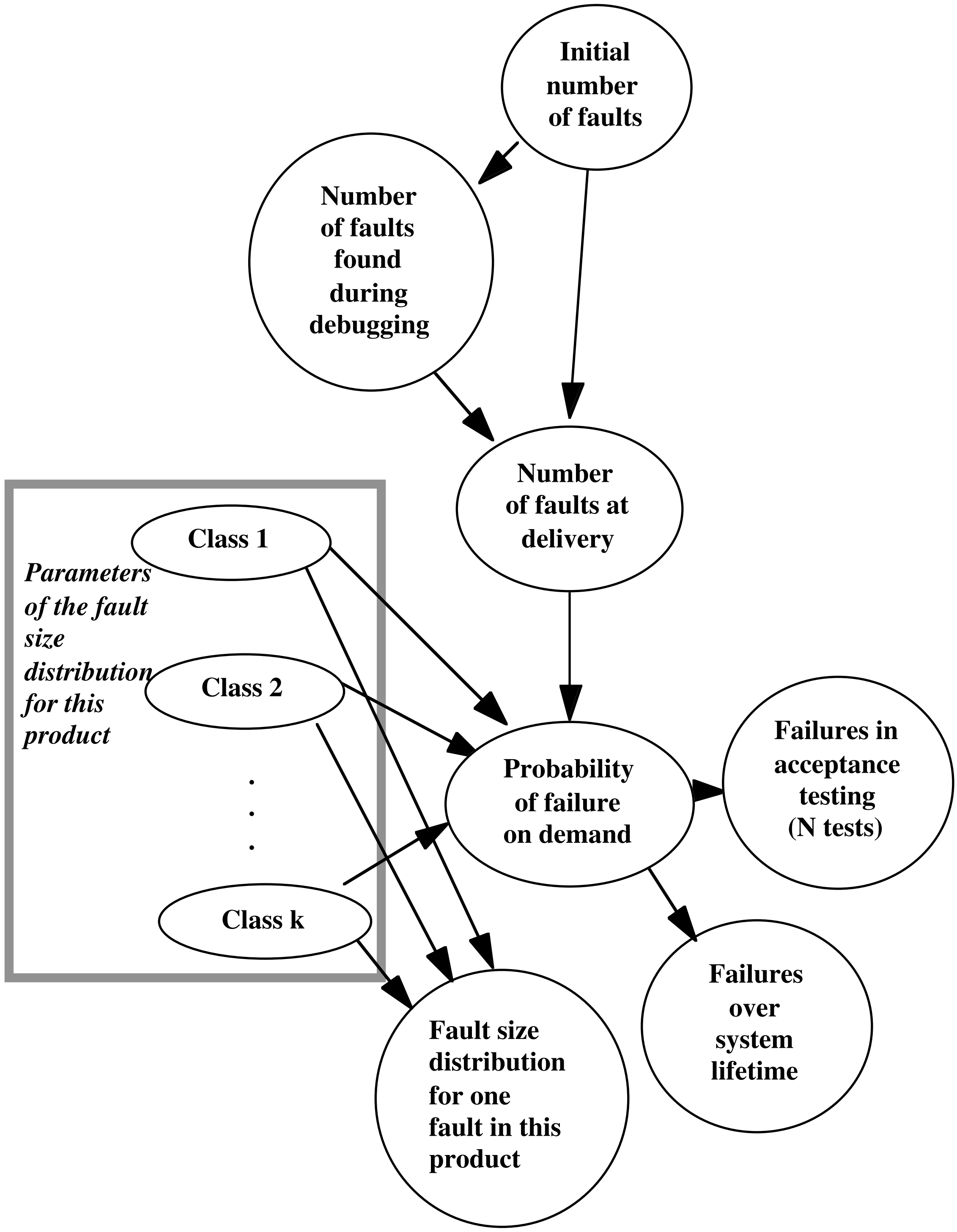


Fig. 6.3. The belief network for our safety case

All the process-related variables in the network are obviously affected by the values of other "explanatory variables" in the process, e.g., the initial number of faults is affected by the size of the product, the inherent complexity of the problem solved, the quality of the development personnel, etc. One could decide to represent this influence by adding parent nodes, representing the product size, problem complexity and quality of personnel, with arcs entering the node representing the initial number of faults. The problem, then, would be how to assign the conditional distributions. One could do this if one had abundant statistical data about projects with different combinations of these three factors (measured on appropriate, standardised scales), or, as an alternative, a clear quantitative model of how these factors affect the number of faults. It is understandable that one would only venture predictions of this sort with some trepidation. If, instead, we choose to reason about a rather homogeneous class of projects, i.e., a class where all the parameters with a known -but difficult to quantify- effect on dependability are the same for all projects, a belief net for products of this class does not need to contain these explanatory variables as explicit nodes.

To derive a distribution of the failure rate from the distribution of the number of faults, one needs some information about the "sizes" of the faults (i.e., for each fault, the probability that it will cause a failure, per demand). Here , we model the uncertainty about this by assuming a population of "possible faults", and that if the software contains, say, *k*, faults, these can be considered as being extracted, independently from this population. This model is akin to the Littlewood model [Littlewood 1980] used for extrapolating software reliability growth. One could conceivably try and infer the sizes to be expected in the faults remaining in the software from the faults found during the debugging phase. After all, if one finds many faults which are larger than usual (without using more powerful fault-finding techniques), one could expect the product to be populated with larger-than-usual faults. We have instead chosen to assume no such link. These assumptions are consistent with a scenario where, in products of the class under considerations, very few faults are usually present in each product, and no special regularity has been observed linking the sizes of the different faults in one product. One can then think that developers insert faults by "picking at random" from a population of available faults. The distribution of sizes in this population can then legitimately be inferred from the samples observed in previous products.

A standard way of modelling this situation for the purpose of allowing inference from new observations is to posit that the distribution of fault sizes (over the population) belongs to a given parametric family of distributions. One can then represent the values of the parameters as nodes in the belief network. These (their "prior" values) would initially be set equal to those measured over the whole population of products. Using observations on this specific project, one would then update the parameter values to represent updated predictions about this individual product.

To describe the distribution of fault sizes, we chose to use parameters that directly represent the values of the probability distribution for different values of the random variable (fault size). More precisely, the meaning of our parameters (corresponding to the nodes in the "cloud" marked "parameters of the fault size distribution for this product") is as follows. To make the modelling easier in intuitive terms, we divide the faults into classes of decreasing sizes. We assume that all fault sizes belong to a succession of discrete values, the negative powers of 10 (or another number that we believe appropriate), so that we call class i the class of all faults with size 10^{-i}. The node "param. i" represents the random variable:

$$\frac{\text{Probability(fault belongs to CLASS}_i)}{\text{Probability(fault does NOT belong to CLASS}_1, \ldots\text{CLASS}_{i-1})}$$

which can also be stated as:

$$\text{Probability(fault belongs to CLASS}_i \mid \text{fault does NOT belong to CLASS}_0, \ldots\text{CLASS}_{i-1})$$

The distributions of these parameters, we repeat, determine the distribution of the faults size for *one* fault chosen at random in this product. This is displayed in the "gauge" node "fault size distribution for one fault in this product", which provides a necessary element of feed-back to the user of the belief network.

One may think that it would be easier for a user to state directly the values of the ordinates for each class, but this would leave the possibility of specifying inconsistent probabilities (e.g. a distribution where the total probability mass is greater than 1). Making the software tool (for manipulating belief networks) able to automatically prevent such inconsistencies would probably be quite useful.

This way of describing a probability distribution is unusual: most people would choose distributions which have some nice mathematical property (in that it simplifies calculations, e.g. allowing the posterior distribution to belong to the same family, and yielding simple, closed-form expressions for the parameters of the posterior distribution). These distributions often also have alternative descriptions in which some of the parameters have a "natural" meaning, like e.g. the expected value of the random variable. In our case, we felt that this kind of "intuitive" parameterisation could be misleading, as the user will be interested in less obvious things, like the probability masses in the "tails" of the distribution.

Clearly, we have constrained the shape of our distribution. By reducing the interval between the sizes of faults in successive classes we could approximate a continuous distributions closely as required. We expect that users will not be able to use a very fine-grained description. The errors that this causes may be kept in check by sensitivity analysis, and we expect them to be minor compared to the other uncertainties involved.

Since the tool we use compels us not to use continuous random variables, we considered using, as an alternative description in terms of a discrete random variable, a continuous "staircase" distribution of fault sizes, with each fault class including faults in a certain interval of sizes. This seemed less intuitive, and its mathematical description more likely to mislead, than the representation we chose.

The distribution of the system Pfd is then obtained from the number of faults after debugging and the parameters describing the distribution of fault sizes. It is a multinomial distribution: given that there are, say, k faults, and given the probabilities of a fault belonging to each of the fault classes, the probability that the Pfd is exactly x is obtained by enumerating all the events in which choosing k faults at random (from the fault classes, with their different sizes of faults) would produce a set of k faults such that its total Pfd is x, and computing the sum of the probabilities of all these random choices. For the total Pfd of a k-tuple we use the sum of the Pfd's of the individual faults. This means that faults do not mask one another and no two faults are ever triggered by the same input, or, equivalently, that the "failure regions" (in the input space) corresponding to different faults are all disjoint. In reality, instead, the failure regions due to different bugs will sometimes overlap, and one could preserve our assumption only by redefining these failure regions: whenever there are two faults, A and B, such that their failure regions overlap, one arbitrarily chooses to split the intersection and assign the two parts one to the failure region of A and one to that of B. If one were trying to model the effects of fault removal (reliability growth) this model would not be satisfactory, since when one faults - say, A- is removed, its failure region disappears *except for* its intersection with the failure region of B. In our case, the error that we may introduce with this model seems minor compared to the overall uncertainty in our predictions.

The node "Failures observed in acceptance testing, in N tests" allows us to update the probability distributions in the belief networks on the basis of test results. The probabilities of observing no failures and of observing some failure in N demands, conditional on the "Probability of failure on demand", is computed as:

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

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

i.e., the outcomes of the demands are considered independent. This represents the expected regime of operation of the protection system (demands being spaced widely apart in time, so that no effect of a demand lingers until the next demand) and of course a sound testing procedure would also preserve this property.

The knowledge modelled by this belief network goes through the following sequence of stages during the project:

- initially, the belief network is populated with probability distributions obtained from the history of previous projects. All the values of the probability distributions are derived from the previous history of similar projects run by the development organisation. The belief network yields a "prior" distribution for the variable of interest, i.e., the probability of failure on demand. Two useful items of information from this distribution are: i) the expected value of the Pfd; ii) the probability of satisfying, or of not satisfying, the requirements. This latter probability is an indication of the risk of the project at this stage;

- when the project (using the organisation's standard methods and decision criteria) 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, then this belief network would no longer represents the available knowledge about this project;

- as debugging proceeds, the number of faults found is logged. At the end of the debugging phase, when the product is deemed ready for final acceptance testing and then delivery, this number is "input to" the belief network as the observed state of the node "Number of faults found during debugging". This changes the distributions for all the nodes in the network, except those describing fault sizes.

  Notice that if the predictions at this point looked worse than the initial predictions, one could decide to perform additional debugging. However, this could not be used to improve the prediction, in that it would be a change to the development process on which the prediction is based. Furthermore, if one tried to input to the belief net the number of faults found with this increased amount of debugging, one could actually obtain a lower predicted dependability (if, as is likely, the number faults found is usually positively correlated with the faultiness of the product).

- acceptance testing is then performed for N demands. If no failures happen (which is expected to be the case), the state of the node "Failures observed in acceptance testing, in N tests" is set to 'no failures", and all the other distributions are updated as a result.

  In successful projects, the state of the node "Failures observed in acceptance testing, in N tests" will be "no failures". Logically, the required number of tests, N, should be set to be at least such that, with the prior distributions for this population of products, the series of failure-free tests demonstrates the required value for "Probability of failure on demand".

- 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 "Failures observed in acceptance testing, in N tests", with a different value of N

Notice also that a requirement in terms of the unobservable random variable "Probability of failure on demand" may look unsatisfactory [Littlewood and Wright 1995], and one may prefer to require, for instance, that a failure (i.e., an observable event) be sufficiently improbable over the lifetime of the system. This is easy to deal with: one can easily add a node representing "number of failures over system lifetime" (as shown at the bottom-right corner of our figure), and possibly an additional one, "number of demands over the lifetime of the system", which would be derived from the safety case of the controlled plant and primary control system, so as to model both the uncertainty about the system Pfd and the uncertainty about the frequency of demands to be expected.

6.2 Use of the model

We derived prior distributions for the nodes of this belief network from a sample of programs developed and tested in a series of academic experiments. We will not discuss in detail the inference procedures used to obtain these distribution from the samples (as a matter of fact, we are still reviewing this part as in some cases we used "naive" but quick procedures). The purpose of this section is to show how these prior distributions can be manipulated and updated. Most of it is obtained directly from screen dumps of the tool we used (HUGIN, distributed by Hugin A/S in Denmark), which show histograms of the probability distributions in each node. The next figure shows how the belief network of figure 6.3 is represented on the screen by HUGIN. We changed the names of the nodes slightly to obtain smaller figures.

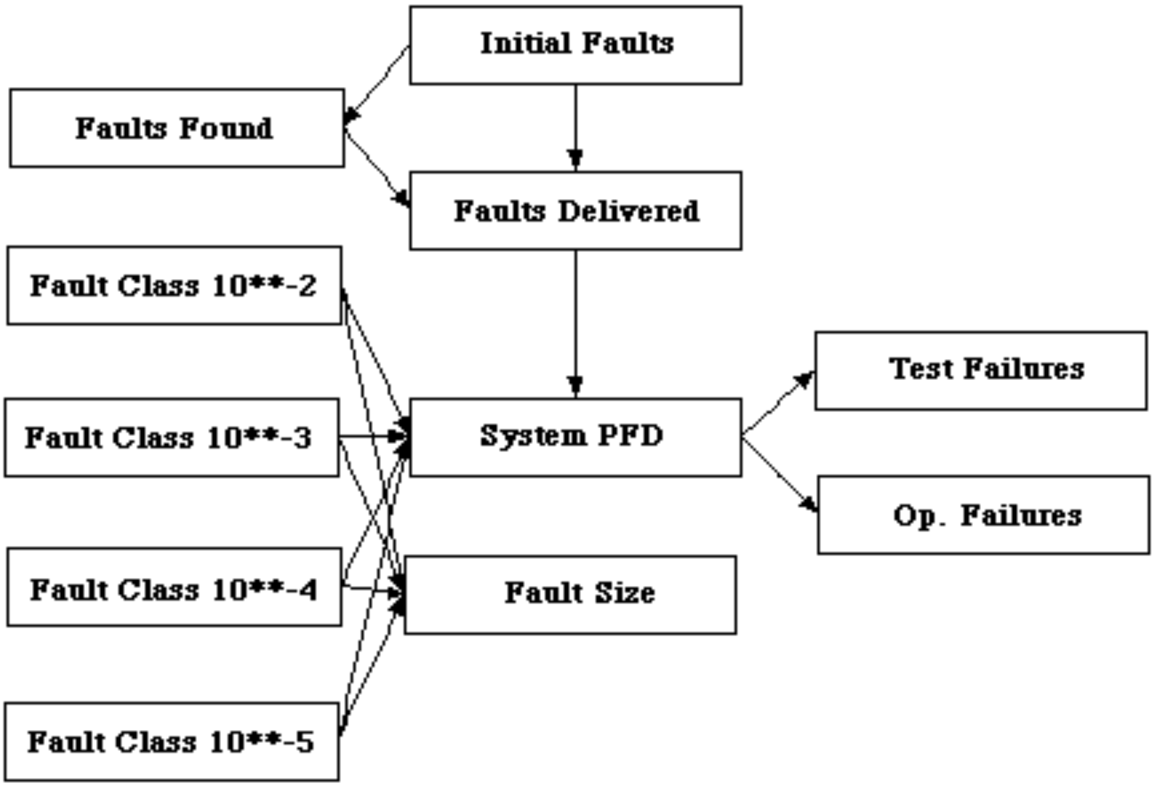


Fig. 6.4. The belief network as represented by the tool we used, with abbreviated names for the nodes (random variables).

We assumed that the acceptance testing simulates 50,000 demands on the system, and that 500 demands are expected over the lifetime of the system.

Prior distributions

The distribution for "Initial faults" was a best fit (to the sample observed) of a binomial distribution with N=16. This rather arbitrary choice would have a serious misleading effect if one were to find and fix 16 faults: this prior distribution would then compel one to conclude that the software is now perfect! We accept this approximation under the assumption that if one were to find more than 8 faults (the maximum ever observed in our sample) one would conclude that the software needs major reworking anyway, and stop the evaluation exercise at this stage. This is one of the many cases in which an argument that seems reasonable in expected circumstances leads to paradoxical conclusions in an unexpected situation. We plan to experiment with different forms of distributions, representing different forms of "plausibility" about the tail of the distribution of the number of faults, as of sensitivity test for the conclusions obtained here.

The number of faults found during debugging was available for a small sub-sample of our sample set. We have not yet tried to solve this inference problem in a rigorous manner. The conditional distribution shown here is, for each value of "Initial faults" a binomial distribution, with N equal to the value of "Initial faults" and p equal to the fraction of faults that were found in debugging over our whole sample. In other words, we assume here 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.

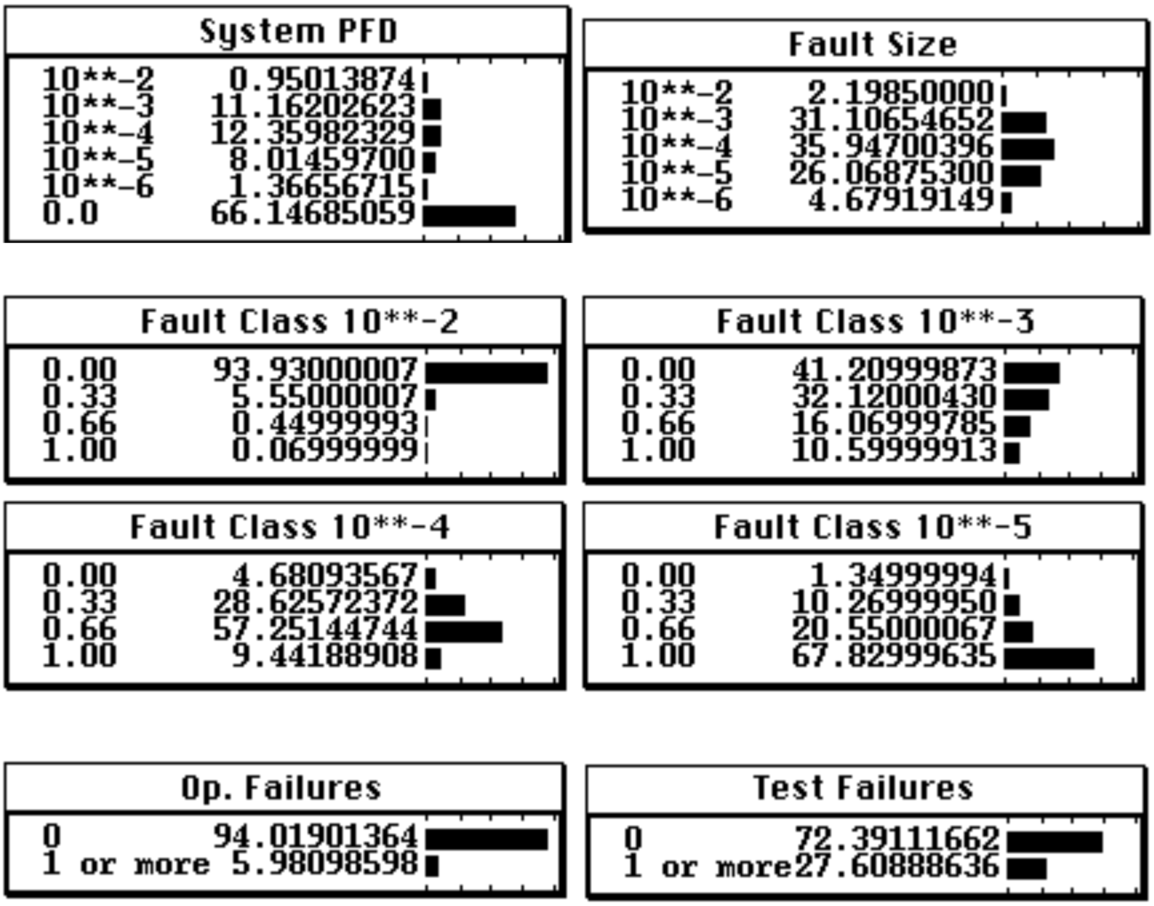
In assigning a conditional distribution to this node, one may obtain the counter-intuitive (but logically correct) effect that the more faults are found, the better the debugged system will appear to be. In this case, our belief that the faults found are an indication of those still present would make us think that the software contains more bugs than usual; but our separate belief in the distribution of "Initial faults" makes us consider it unlikely that many faults are present, so that, the more faults we have found, the fewer are likely to remain. The interplay between these distributions decides whether we consider finding many faults an indication of good debugging or of bad software as well, and whether we decide that an unusually effective debugging phase outbalances the initial high number of faults.

Initial Faults	
0	17.88221747
1	32.49863982
2	27.68536806
3	14.67509568
4	5.41736558
5	1.47680519
6	0.30753072
7	0.04990158
8	0.00637662
9	0.00064382
10	0.00005119
11	0.00000317
12	0.00000015
13	0.00000001
14	0.00000000
15	0.00000000
16	*

Faults Found	
0	27.98919380
1	37.09667027
2	23.04733545
3	8.90947357
4	2.39861161
5	0.47686528
6	0.07242053
7	0.00857015
8 or more	0.00086104

Faults Delivered	
0	66.14684463
1	27.69417763
2	5.43511063
3	0.66370247
4	0.05644364
5	0.00354474
6	0.00017005
7	0.00000636
up to 16	0.00000019

We assigned the distributions for the nodes "System Pfd" and "Fault Size" as explained in Section 6.1



Comments

These prior distributions have some interesting implications. The expected value for "System Pfd" is about 2 10^{-4}.

Another interesting prediction is the probability of the project failing to deliver software with the desired value of Pfd. If, for instance, the requirement is Pfd<10^{-3}, the probability that the software will not satisfy the requirement, i.e., Pfd $\geq 10^{-3}$, is about 12 %. Even more worrisome for the developers is the fact that the product has a 28 % probability of not passing its acceptance test. The 6 % probability of at least one failure (on demand) over the system lifetime is also worrisome, of course. However, the public and the licensing authorities would be protected because the system will not be deployed unless it passes its acceptance test (50,000 runs) first.

Inference from acceptance testing only

We may then consider whether this "average" product would be certifiable with some confidence, *assuming that it passed the prescribed acceptance testing*. This means asking what the probabilities would look like after setting the value of the node "Test failures" to 0.

This is shown in the following figures.

Initial Faults	
0	24.70222861
1	35.97114086
2	24.69639629
3	10.60168520
4	3.18236314
5	0.70778495
6	0.12058084
7	0.01604392
8	0.00168437
9	0.00008687
10	0.00000423
11	0.00000006
12	0.00000000
13	0.00000000
14	0.00000000
15	0.00000000
16	*

Faults Found	
0	27.25277841
1	36.90728545
2	23.42996895
3	9.25537720
4	2.54630633
5	0.51733609
6	0.08029402
7	0.00971119
8 or more	0.00094046

Faults Delivered	
0	91.37426019
1	7.84430206
2	0.76448307
3	0.01615933
4	0.00075936
5	0.00002851
6	0.00000089
7	0.00000002
up to 16	0.00000000

System PFD	
10**-2	*
10**-3	*
10**-4	0.11501339
10**-5	6.71503171
10**-6	1.79568827
0.0	91.37426615

Fault Size	
10**-2	2.12226920
10**-3	29.71839905
10**-4	35.61927974
10**-5	27.45051980
10**-6	5.08952662

The expected value of the System Pfd is now about 8 10^{-7}, and the probability of Pfd ≥ 10^{-3} has fallen to about 0 (that is, lower than the lowest number represented by our tool). In other words, products of this category, with this amount of acceptance testing, are a much lesser risk for the regulator than for the developer: once the product has passed its acceptance testing, there is a very good probability that it really satisfies is requirements. Another appropriate comment is that in this case an adequate amount of testing provides much stronger evidence of reliability than offered by "process" considerations.

Op. Failures	
0	99.95999932
1 or more	0.03999843

Test Failures	
0	100.00000000
1 or more	-

We do not show, for brevity, the distributions for the "Fault Class" nodes. The expectation for the future operational life of the system is that there will be failures with a probability of about 4 10^{-4}. This, of course, is a direct implication of the Pfd distribution. It is worth pointing out that if the requirement is indeed Pfd< 10^{-3}, and 500 demands are expected over the lifetime of the system, then the requirements only imply that the probability of having some failure during operation is less than about 0.4. A Pfd of 10^{-4} would imply a probability of failures of about 0.05.

Inference from debugging and acceptance testing

We now consider how the developers' expectations would be changed by observations during the project. Assume, for instance, that they find and correct 2 faults, and then the software passes the 50,000 tests. The posterior distributions that this generates are shown below.

Initial Faults	
0	*
1	*
2	91.81546569
3	7.48443231
4	0.68600886
5	0.01348789
6	0.00058200
7	0.00001976
8	0.00000055
9	0.00000001
10	0.00000000
11	0.00000000
12	0.00000000
13	0.00000000
14	0.00000000
15	0.00000000
16	*

Faults Found	
0	-
1	-
2	100.00000000
3	-
4	-
5	-
6	-
7	-
8 or more	-

Faults Delivered	
0	91.81546569
1	7.48443305
2	0.68600886
3	0.01348789
4	0.00058200
5	0.00001976
6	0.00000055
7	0.00000001
up to 16	0.00000000

System PFD	
10**-2	*
10**-3	*
10**-4	0.10831794
10**-5	6.36818036
10**-6	1.70803368
0.0	91.81546569

Fault Size	
10**-2	2.12636925
10**-3	29.79161441
10**-4	35.63847244
10**-5	27.37604082
10**-6	5.06750420

Fault Class 10**-2	
0.00	94.11219358
0.33	5.39828166
0.66	0.42525060
1.00	0.06427082

Fault Class 10**-3	
0.00	42.86311269
0.33	32.02770650
0.66	15.37674069
1.00	9.73243862

Fault Class 10**-4	
0.00	5.16289696
0.33	30.22743165
0.66	55.92269301
1.00	8.68697539

Fault Class 10**-5	
0.00	1.39924185
0.33	10.49278006
0.66	20.70602328
1.00	67.40195155

Op. Failures	
0	99.96209741
1 or more	0.03789813

Test Failures	
0	100.00000000
1 or more	-

The expected value of the Pfd is still around 8 10^{-7}, though slightly less (by about 5 %) than it would be without having found these two faults. The other prediction do not change by much either. We point out that 2 faults is very close to the number of faults we would expect to find on the basis of the prior distribution. So, the debugging process brought no surprises and no reason for reviewing our initial expectations.

7. Discussion

This report shows how belief networks can be used to describe a complex argument, and then, with the aid of a software tool, to explore its consequences and update it with the results of new observations. We have first presented some very simple examples to show the power of this formalism and calculus, and then described a more realistic example which could be part of an actual safety case.

Our treatment of this example has been rather incomplete, for the sake of brevity. In an actual use of this belief network for a safety case, a full, detailed discussion would be necessary for all the assumptions used, showing that they do represent the available knowledge of the experts and are either well-justified or acceptable as they do not introduce dangerous errors in the conclusions. Likewise, the conclusions that can be drawn from the network and the available measurements would be explored in detail, to look for inconsistencies or other unexpected conclusions. More importantly, this process of building the network and studying it would be likely to be iterative, as one would not expect to produce the "right" network at the first attempt, but to use the networks one initially propose 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; however, the claim we make for this method is that it would make safety assessment exercises more trustworthy, not less boring.

Our conclusion so far is that the method is indeed useful. "Playing" with our example allowed us to reason about how we were building our argument, which parts of our intuitions were right and which were wrong, which evidence was really useful. We believe that this method should be applied to real-life case studies, to see whether its usefulness is confirmed.

One can make serious errors in applying this method, like any other probabilistic methods. The problem is that the safety case is in fact a complex design, leading to a complex computer algorithm. The fact that the complex details are, once input to the computer, hidden behind a visually intuitive user interface tempts one to gloss over those details, and produce a "Garbage in, garbage out" situation. The temptation to stop at the first belief net that seems to support one's claims will be strong. A belief net produced in support of a safety case should be seen not as irrefutable evidence, but as a support that makes the safety case visible and auditable.

There is a clear need to improve the software tools available. Probably because they were developed as expert system shells, the tools we tried had insufficient capabilities of displaying the probability distributions in a user-friendly manner and of dealing with the minuscule probabilities needed in safety assessment. Further help for use in safety assessment 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).

At this stage, we believe that investment in such tools would be rewarded by a substantial improvement in the ability of assessors to understand, refine and audit "engineering judgement" on complex safety cases. 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?" From the point of view of certification practice, 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.

References

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

[CACM 1995] CACM, "Real-World Applications of Bayesian Networks", *Communication of The ACM, Special Issue*, 38 (3), pp.24-57, 1995.

[Charniak 1991] E. Charniak, "Bayesian Networks without Tears", *AI Magazine*, Winter 1991, pp.50-63, 1991.

[Cheeseman 1988] P. Cheeseman, "An inquiry into computer understanding", *Computational Intelligence*, 4 (57), pp.58-66, 1988.

[Cooper and Herskovits 1991] G. F. Cooper and E. Herskovits, "A Bayesian Method for Constructing Bayesian Belief Networks from Databases", in *Uncertainty in AI,* pp.86-94, 1991.

[Dubois and Prade 1988] D. Dubois and H. Prade, *Possibility Theory - An Approach to computerized Processing of Uncertainty,* Plenum Press, New York, 1988.

[Fenton 1993] N. Fenton, "How effective are software engineering methods?", *Journal of Systems and Software*, 20, pp.93-100, 1993.

[Fenton *et al.* 1994] N. Fenton, S. Pfleeger and R. Glass, "Science and Substance: A Challenge to Software Engineers", *IEEE Software*, pp.86-95, July 1994.

[Feynman 1988] R. Feynman, *What do you care what other people think? - further adventures of a curious character,* Unwin Hyman, London, 1988.

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

[Henrion *et al.* 1991] M. Henrion, J. S. Breese and E. J. Horvitz, "Decision Analysis and Expert Systems", *AI Magazine*, 12 (4), pp.64-91, 1991.

[Herskovits 1994] *ERGO - Demo Program and Documentation*, Noetic Systems Incorporated - Baltimore, Demo, 1994.

[Jensen 1993] F. V. Jensen, "Introduction to Bayesian Networks", in *Hugin Docs* HUGIN, Aalborg, Denmark, 1993.

[Kailar *et al.* 1994] R. Kailar, V. D. Gligor and L. Gong, "On the Security Effectiveness of Cryptographic Protocols", in *4-th international Working Conference on Dependable Computing for Critical Applications, DCCA-4,* (San Diego), pp.90-101, 1994.

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

[Lam and Bacchus 1994] W. Lam and F. Bacchus, "Learning Bayesian Belief Networks : An Approach Based on MDL Principle", *Computational Intelligence*, pp.269-93, 1994.

[Lauritzen and Spiegelhalter 1988] S. L. Lauritzen and D. J. Spiegelhalter, "Local Computations with Probabilities on Graphical Structures and Their Application to Expert Systems - with discussion-", *Journal of Royal Statistical Society, Series B*, 50 (2), pp.157-224, 1988.

[Leveson 1991] N. G. Leveson, "Software Safety in Embedded Computer Systems", *CACM*, 34 (2), pp.34-46, 1991.

[Littlewood 1980] B. Littlewood, "Theories of Software Reliability: How good are they and how can they be improved?", *IEEE Transactions on Software Engineering*, SE-6 (5), pp.489-500, 1980.

[Littlewood and Strigini 1993] B. Littlewood and L. Strigini, "Validation of Ultra-High Dependability for Software-based Systems", *Communications of the ACM*, 36 (11), pp.69-80, 1993.

[Littlewood and Wright 1995] B. Littlewood and D. Wright, "A Bayesian model that combines disparate evidence for the quantitative assessment of system dependability", in *SafeComp95,* (Belgirate, Italy), Springer, 1995.

[Ng and Abramson 1990] K.-C. Ng and B. Abramson, "Uncertainty Management in Expert Systems", *IEEE Expert*, April, pp.129-42, 1990.

[Olesen and al 1994] K. G. Olesen and e. al, *aHUGIN: A System Creating Adaptive Causal Probabilistic Networks*, ODIN Research Group, 1994 1994.

[Pearl 1986] J. Pearl, "Fusion, Propagation and Structuring in Belief Networks", *Artificial Intelligence*, 29, pp.241-88, 1986.

[Pearl 1988] J. Pearl, *Probabilistic Reasoning in Intelligent Systems,* Morgan Kaufmann, Palo Alto, CA, 1988.

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

[Powell 1992] D. Powell, "Failure Mode Assumptions and Assumption Coverage", in *Proc. 22nd International Symposium on Fault-Tolerant Computing,* (Boston, Massachusetts, USA), pp.386-95, 1992.

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

[Saffiotti *et al.* 1992] A. Saffiotti, S. Parsons and E. Umkehrer, *A Case Study in Comparing Uncertainty Management Techniques*, Universite Libre de Bruxelles, Technical Report, N°TR/IRIDIA/92-28, 1992 1992.

[Strigini 1994a] L. Strigini, "Considerations on current research issues in software safety", *Reliability Engineering and System Safety*, 43 (2), pp.177-88, 1994a.

[Strigini 1994b] L. Strigini, *Engineering judgement in reliability and safety and its limits: what can we learn from research in psychology?*, SHIP project, Technical Report, N°T/030, July 1994b.

[Wright 1995] D. Wright, *Application of Graphical Probabilistic Networks*, SHIP project, Technical report, N°SHIP/T/044, May 1995.

[Wright and Cai 1994] D. Wright and K.-Y. Cai, *Representing Uncertainty for Safety Critical Systems*, SHIP Project, Technical report, N°SHIP/T/002, May 1994 1994.

[Zadeh 1975] L. A. Zadeh, "The concept of a linguistic variable and its application to approximate reasoning", *Information Sciences*, 1975 (1 & 2), pp.199-249, 301-57, 1975.

[Zadeh 1978] L. A. Zadeh, "Fuzzy sets as a basis for a theory of possibility", *Fuzzy Sets & Systems*, 1, pp.26(3-8), 1978.