European Journal of Computer Science and Information Technology
Vol.2, No.1, pp.30-44, March 2014
Published by European Centre for Research Training and Development UJK (www.ea-journals.org)

Predicting Software Reliability and Defects Using Bayesian Networks

Matthias Daniel, Ette Harrison Etuk
Dept. Mathematics and Computer Science, Rivers State University of Science and Technology, Nigeria

ABSTRACT: *This paper reviews the use of Bayesian networks (BNs) in predicting software reliability and software defects. The approach allows analysts to incorporate causal process factors as well as combine qualitative and quantitative measures, hence overcoming some of the well known limitations of traditional software metrics methods. The approach has been used and reported on by organizations such as Motorola, Siemens, and Philips. However, one of the impediments to more widespread use of BNs for this type of application was that, traditionally,BN tools and algorithms suffered from an obvious 'Achilles' heel' – they were not able to handle continuous nodes properly, if at all. This forced modelers to have to predefine discretization intervals in advance and resulted in inaccurate predictions where the range, for example, of defect counts was large. Fortunately, recent advances in BN algorithms now make it possible to perform inference in BNs with continuous nodes, without the need to pre-specify discretization levels. Using such 'dynamic discretization' algorithms results in significantly improved accuracy for reliability and defects prediction type models*.

KEYWORDS: Software, Reliability, Defects, Bayesian networks, Predicting

INTRODUCTION

In 1993 at what was then the leading international software metrics conference Applications of Software Metrics ASM 93 (in La Jolla, California), a leading metrics expert recounted an interesting story about a company-wide metrics programme that he had been instrumental in setting up. He said that one of the main objectives of the programme was to achieve process improvement by learning from metrics what process activities worked and what ones did not. To do this the company looked at those projects that, in metrics terms, were considered most successful. These were the projects with especially low rates of customer-reported defects, measured by defects per thousand lines of code (KLOC). The idea was to learn what processes characterized such successful projects. A number of such 'star' projects were identified, including some that achieved the magical perfect reliability target of zero defects per KLOC in the first 6 months post-release. However, it turned out that what they learned from this was very different to what they had expected. Few of the star projects were, in fact, at all successful from any commercial or subjective perspective. The main explanation for the very low number of defects reported by customers was that they were generally so poor that they never got properly completed or used. Therein lies the classic weakness of traditional software metrics that has been highlighted in [1] and [2] – the omission of sometimes obvious and simple causal factors that can have a major explanatory effect on what is observed and learnt. If during a software development project it is reported that very few defects were found during a critical testing phase is that good news or bad news? Of course it depends on the testing effort, just as it does if it was reported that a large number of defects were discovered. Ideally, any model using defects found in one phase of testing to

predict defects found in subsequent phases (whether by testing in-house or by customers post-release) should incorporate causal factors such as testing effort and process quality. Bayesian networks (BNs) are causal models that enable this to be done. The earliest work that explicitly attempted to use such models and information about intermediate software products to predict later features of software quality was Hall et al. in 1992 [3].

In 1996 Neil and Fenton [4] proposed explicitly how BNs could be used for defect prediction, and there were related contributions in [5–9]. Until very recently progress in this type of modeling was hampered by both the limitations of the BN modeling tools and difficulties in eliciting the necessary information for realistic models. This paper reviews the recent progress in the former (the latter has been dealt with in [10]). The basis of the BN solution to prediction defect by means of a very simple BN is explained in section 2, and in section 3 the kind of commercial-scale BN models that have been implemented by organizations such as Motorola, Siemens, and Philips are highlighted. In section 4 the solution of one of the major weakness of the BN models is discussed. The weakness is the well-known 'Achilles' heel' of BN inference algorithms; specifically, their inability to properly handle continuous (non-Gaussian) variables. Recent work on dynamic discretization algorithms [11] has largely fixed this 'Achilles' heel' and, in the case of reliability and defect prediction, results in significantly increased accuracy.

2 SIMPLE CAUSAL MODEL FOR PREDICTION SOFTWARE DEFECT

A BN is a directed graph (such as that shown in Fig. 1) together with a set of probability distributions. The directed graph is referred to as the 'qualitative' part of the BN, while the probability distributions are referred to as the 'quantitative' part. In the qualitative part the nodes represent uncertain variables and the arcs represent the existence of a causal/influential relationship between two variables.

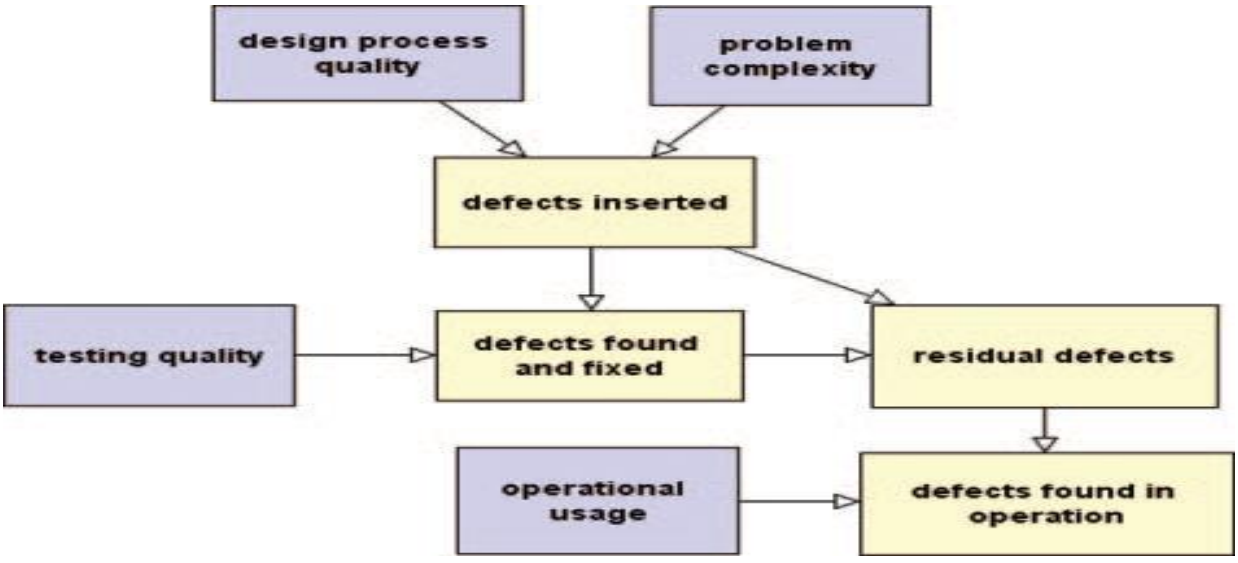

Fig. 1 Simplified version of a BN model for software reliability and defects prediction

In this example the number of defect found in operation (i.e. those found by customers) in a software module is what is of interest to predict. This factor is dependent on the number of residual defects. However, it is also critically dependent on the amount of operational usage. If the system is not used then no defects will be found irrespective of the actual number of defects. The number of residual defects is determined by the number introduced during development minus the number that are successfully found and fixed. Obviously the number of defects found and fixed is dependent on the number of defects introduced into the system. The number introduced into the system is influenced by problem complexity and design process quality. The better the design the fewer the number of defects and the less complex the problem the fewer the number of defects. Finally, the number of defects found is influenced not just by the number in there to find but also by the level of the testing effort.

For the quantitative part of the BN there is, associated with each child node, a conditional probability distribution (CPD). Nodes without parents are quantified through their marginal probability distributions. If the set of possible values of a node and its parents are all discrete and finite then the CPD for the node can simply be a table that specifies the probability of each discrete state of the node given each combination of states of its parents. In this case the CPD is called a Node Probability table (NPT) and might look something like the one shown in Fig. 2.

Testing Quality	Poor			Good		
Defect Inserted	0	1	2	0	1	2
0	1.0	0.9	0.8	1.0	0.1	0.0
1	0.0	0.1	0.2	0.0	0.9	0.1
2	0.0	0.0	0.0	0.0	0.0	0.9

Fig. 2 NPT for the node ‘defects found’ in the case of simple discrete states

In this example if it is known that two defects were inserted and that the testing quality is ‘good’ then the probability of finding the two defects is 0.9, whereas the probability of finding just one defect is 0.1. It is clear even from the simple model in Fig. 1 that in many situations the CPD needs to be defined as a function rather than as an exhaustive table of all potential parent state combinations. Some of these functions are deterministic rather than probabilistic: for example, the ‘residual defects’ is simply the numerical difference between the ‘defects inserted’ and the ‘defects found and fixed’. In other cases, standard statistical functions can be used. For example, in this simple version of the model it is assumed that ‘defects found and fixed’ follow a binomial B(n,p) distribution where n is the number of defects inserted and p is the probability of finding and fixing a defect (which in this case is derived from the ‘testing quality’); in more sophisticated versions of the model the p variable is also conditioned on n to reflect the increasing relative difficulty of finding defects as n decreases. Table 1 lists the full set of conditional probability distributions for the nodes (that have parents) of the BN model of Fig. 1. Note that the nodes ‘design quality’, ‘complexity’, ‘testing quality’, and ‘operational usage’ are all ranked nodes in the sense of [10] which means that they have an underlying [0,1] scale that enables them to be used in functions as described in Table 1. The nodes without parents are all assumed to have a prior uniform distribution, i.e. one in which any state is equally as likely as any other state (in the ‘real’ models the distributions for

Table 1 CPDs for the nodes of the BN model in Fig. 1

Node name	CPD
Defects found in Operation	Binomial (n, p) where n = 'residual defects' and p = 'operational usage'
Residual defects	Defects inserted – Defects found (and fixed) in testing
Defects found in Testing	Binomial (n, p) where n = 'defects inserted' and p = 'testing quality'
Defects inserted	Truncated normal with range of zero to 500 with mean complexity x (1-design) x 90 and variance 300

such nodes would normally not be defined as uniform but would reflect the historical distribution of the organization either from data or expert judgment).

The BN is a compact representation of the complete joint probability distribution of all the variables. The power of BN models comes with the fact that, as 'evidence' is entered (meaning that specific values are assigned to variables in the model) the joint probability distribution can be recalculated conditioned on this 'evidence' using Bayesian propagation. The updated marginal probability distribution of each variable can then be observed. There are many tools that perform the necessary calculations. In the remainder of this paper the use of BN calculations will be illustrated and it will be highlighted that using BNs as causal models for software reliability and defects prediction is simple and compelling.Figure 3 shows the marginal distributions of the simple model before any evidence has been entered. Thus, this represents the uncertainty before any specific information is entered about this module. Since uniform distributions for nodes without parents is assumed it is clear that the module is just as likely to have a very high complexity as a very low one, and that the number of defects found and fixed in testing is in a wide range where the median value is

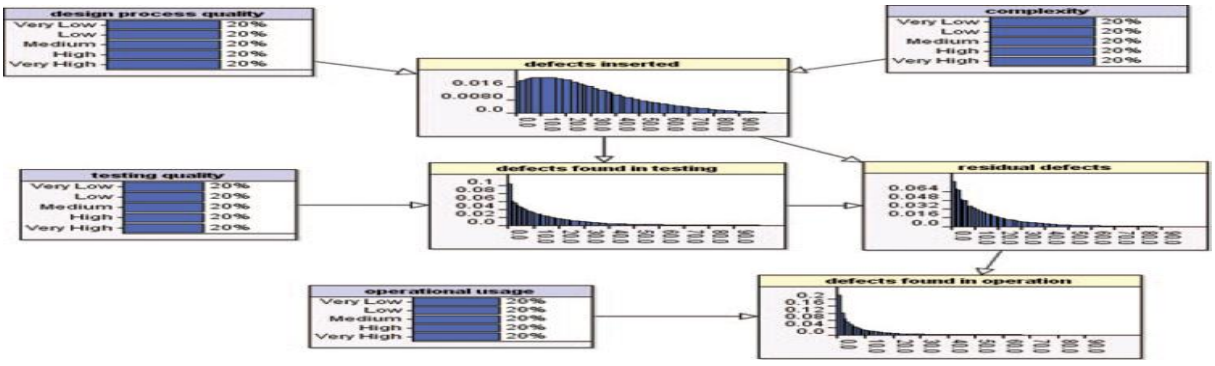

Fig. 3 BN model with marginal distributions for variables superimposed on nodesabout 20–22 (the prior distributions here were for a particular organization's modules). Figure 4 shows the result of entering four observations about this module:

(a) that it had zero defects found and fixed in testing;
(b) that the problem complexity is 'high';
(c) that design process quality is 'medium';
(d) that the operational usage is 'very high'.

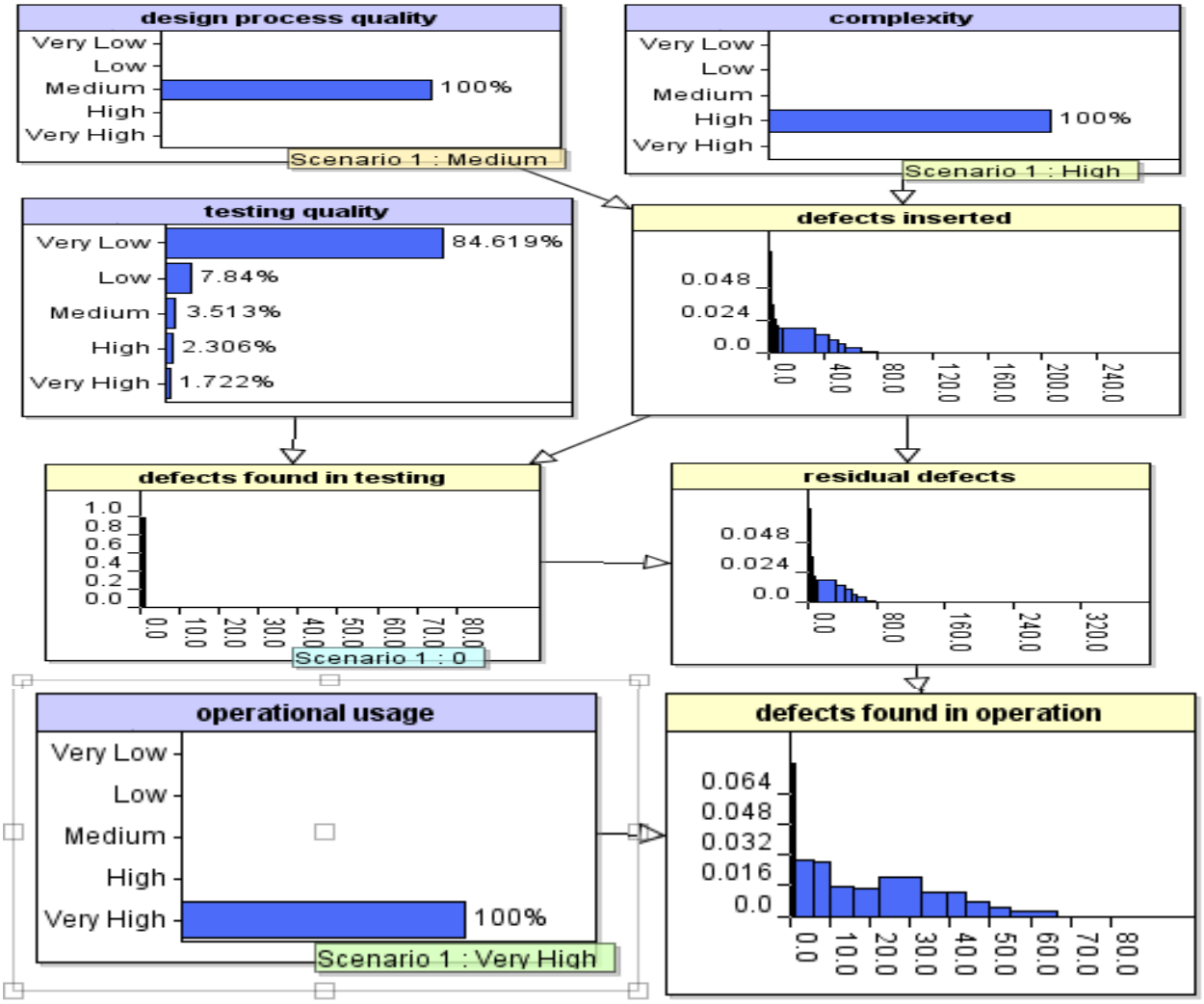

Fig. 4 Zero defects in testing, medium design process quality, high complexity, and very high operational usage observed. Note that all the other probability distributions are updated. The predicted number of operational defects is high (the mean of the distribution being around 22). If it is confirmed that testing quality is 'very low' then when you enter this observation the predicted number of operational defects increases to a mean of 25. But, suppose we discover that the testing effort was actually 'very high'. Entering this observation, the prediction for operational defects drops dramatically to close to a mean of 0. Suppose we find a high number of defects in testing, say 35 (Fig. 5). The predicted number of operational defects has a mean close to 1. Why? Because

a) we tested out most of the defects, and
b) the operational usage is low.

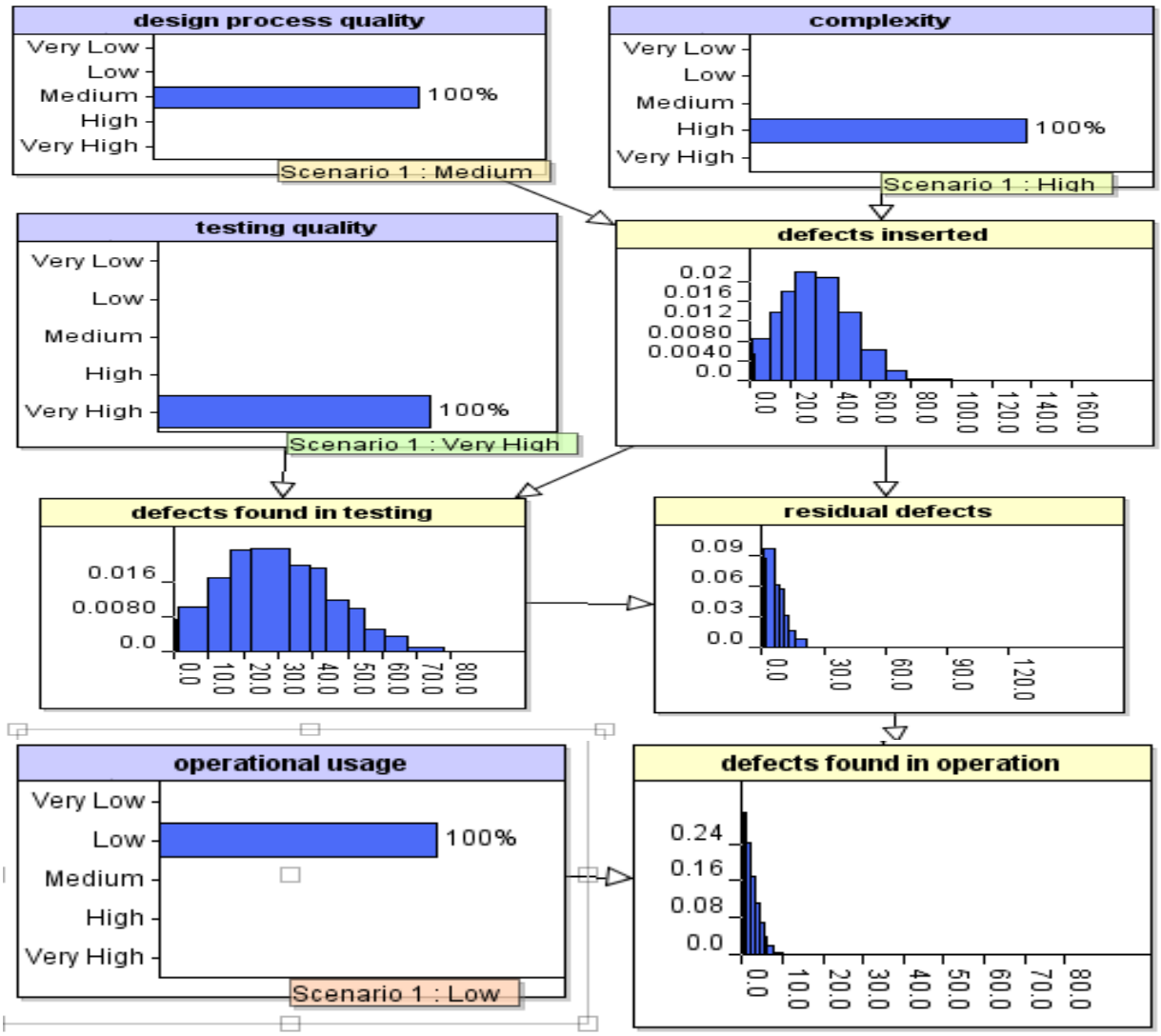

Fig. 5 Very high testing quality, medium design process quality, high complexity, and low operational usage observed

The entering of zero(0) for defects found in operation (Fig. 6). The most likely explanation is that there is very low operational usage. But note also that it is likely that testing and process quality are higher than average and problem complexity is lower than average. Suppose we discover that, in fact, the operational usage is 'medium' and that a rather high number, 30, defects were found in testing.

After calculating the model is more or less convinced that the explanation must be that the testing quality was only 'medium' then the model starts to believe that low complexity and high design process quality are the reason for the result. This type of reasoning is unique to BNs. It provides a means for decision makers (such as quality assurance managers in this case) to make decisions and interventions dynamically as new information is observed.

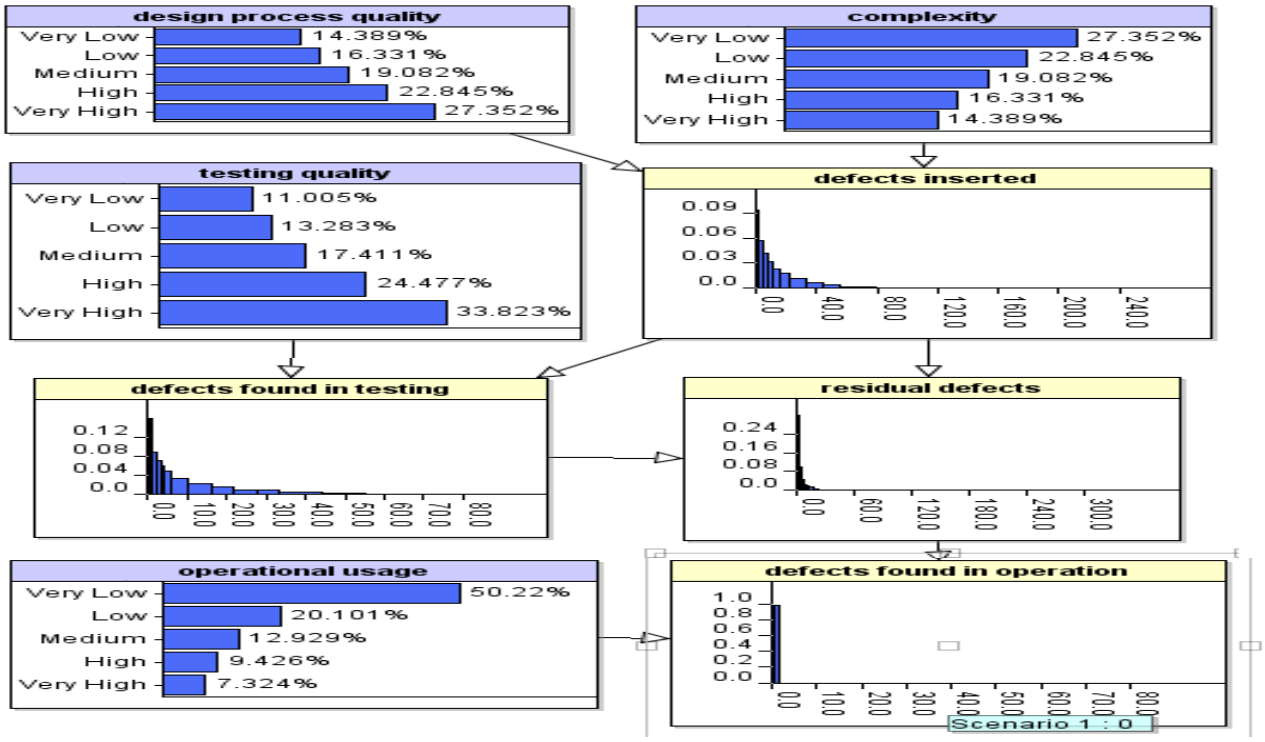

Fig. 6 Zero defects found in operation observed

3 THE DEFECT PREDICTION MODELS FOR COMMERCIAL-SCALE VERSIONS

The ability to do the kind of prediction and what–if analysis described in section 2 has proved to be very attractive to organizations who need to monitor and predict software reliability and defects, and who already has collected defect-type metrics. Hence, organization such as Motorola [2, 14], Siemens [15], and Philips [16] have exploited models and tools originally developed in [17] to build large-scale versions of the kind of model described in section 2. It is beyond the scope of this paper to describe the details of these models and how they were constructed and validated, but what typifies the approaches is that they are based around a sequence of testing phases, including testing activities such as system testing, integration testing, and acceptance testing that are defined as part of the companies' software processes (and hence for which relevant defect and effort data is formally recorded). In some cases a testing phase is one that does not involve code execution, such as design review.

The final testing phase is generally assumed to be operational testing which normally takes the form of some fixed period of post-release testing; it is information on this final phase that enables the organizations to monitor and predict reliability. Corresponding to each phase is a 'subnet' such as the one in Fig. 7, where a subnet is a component of the BN with interface nodes to connect the component subnet to other parent and child subnets. For the final 'operational' phase, there is, of course, no need to include the nodes associated with defect fixing and insertion.

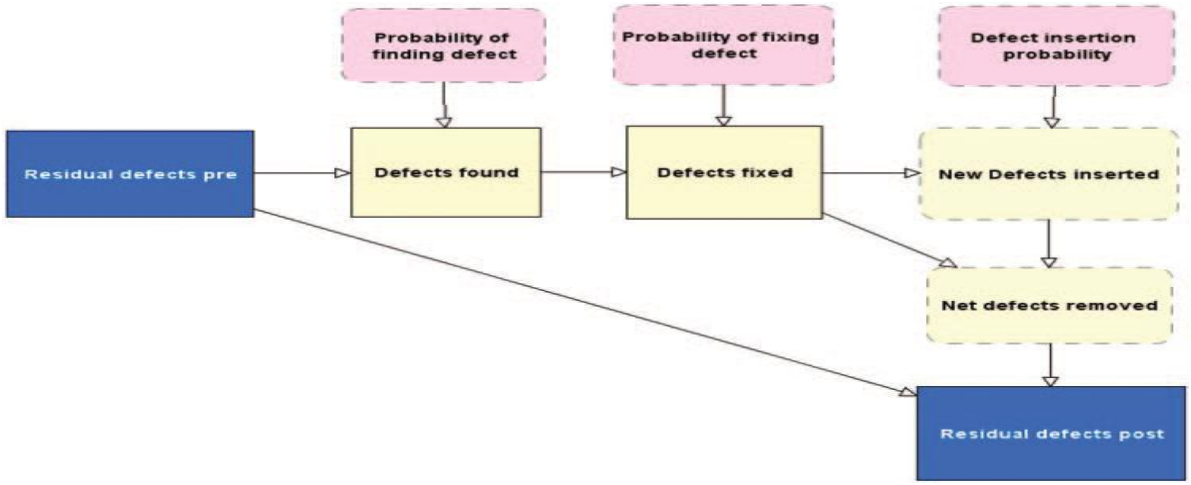

Fig. 7 Defects phase subnet

The distributions for nodes such as 'probability of finding defect' derive from other subnets such as the one shown in Fig. 8. The particular nodes and distributions will, of course, vary according to the type of testing phase.

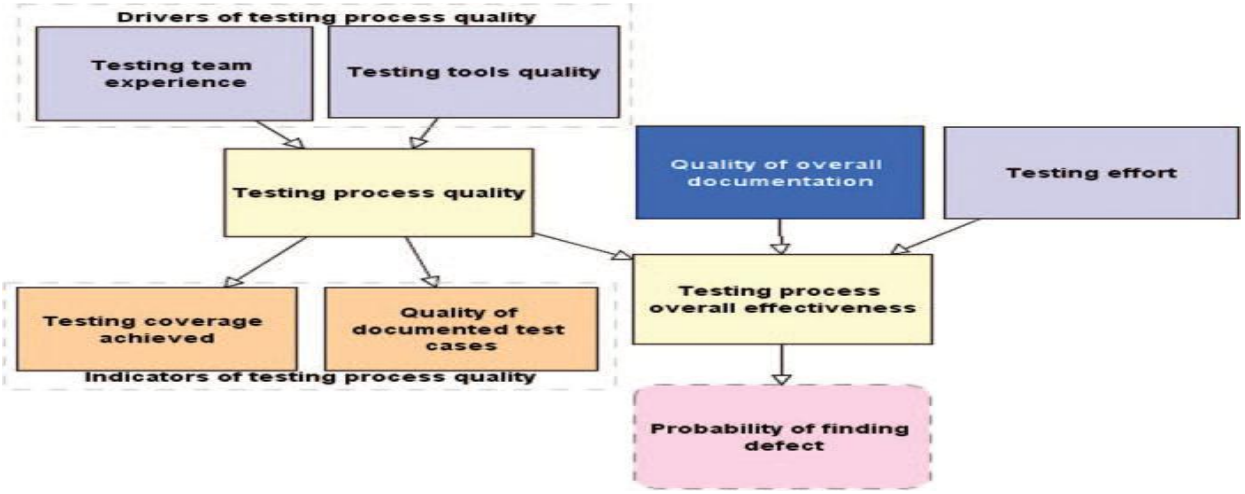

Fig. 8 Typical subnet for testing quality

Two examples are now presented to give a feel for the kind of expert elicitation and data that was required to complete the CPDs in these kinds of models. These examples are the nodes 'probability of finding a defect' and 'testing process overall effectiveness'

1. The CPD for the node 'probability of finding a defect'. This node is a continuous node in the range [0,1] that has a single parent 'testing process overall effectiveness' that is a ranked node (in the sense of [10] on a five-point scale from 'very low' to 'very high'). For a specific type of testing phase (such as integration testing) the organization had both data and expert judgment that enabled them to make the following kinds of assessment.

> Typically (i.e. for our average level of test quality) this type of testing will find approximately 20 per cent of the residual defects in the system. At its best (i.e. when the level of testing is at its best) this type of testing will find 50 per cent of the residual defects in the system; At its worst it will only find 1 per cent

Based on this kind of information the CPD for the node 'probability of finding a defect' is a partitioned expression such as the one in Table 2. Thus, for example, when overall testing process effectiveness is average, the probability of finding a defect is a truncated normal distribution over the range [0,1] with mean 0.2 and variance 0.001.

Table 2 CPD for node 'probability of finding a defect'

Parent (overall testing process effectiveness) state	Probability of finding a defect
Very low	TNormal (0.01, 0.001, 0, 1)
Low	TNormal (0.1, 0.001, 0, 1)
Average	TNormal (0.2, 0.001, 0, 1)
High	TNormal (0.35, 0.001, 0, 1)
Very high	TNormal (0.5, 0.001, 0, 1)

2.The CPD for the node 'testing process overall effectiveness'. This node is a ranked node on a five-point ranked scale from 'very low' to 'very high'. It has three parents 'testing process quality', 'testing effort', and 'quality of overall documentation' which are all also ranked nodes on the same five-point ranked scale from 'very low' to 'very high'. Hence, the CPD in this case is a table of 625 entries. Such a table is essentially impossible to elicit manually, but the techniques described in [10] (in which ranked nodes are mapped on to an underlying [0,1] scale) enabled experts to construct a sensible table in seconds using an appropriate 'weighted expression' for the child node in terms of the parents. For example, the expression elicited in one case was a truncated normal (on the range [0,1]) with mean equal to the weighted minimum of the parent values (where the weights were: 5.0 for 'testing effort', 4.0 for 'testing quality', and 1.0 for 'documentation quality') and the variance was 0.001. Informally this weighted minimum expression captured expert judgment such as

> 'documentation quality cannot compensate for lack of testing effort, although a good testing process is important'.

As an illustration Fig. 9 shows the resulting distribution for overall testing process effectiveness when testing process quality is average, quality of documentation is very high, but testing effort is very low.

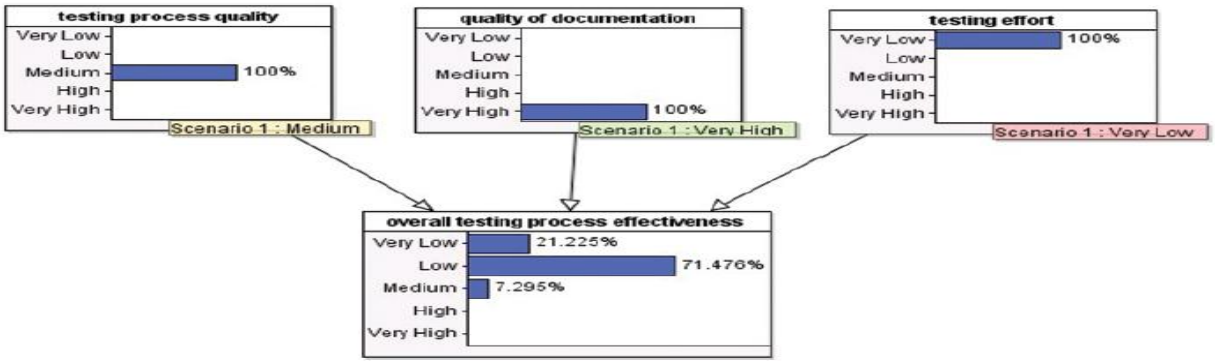

Fig. 9 Scenario for 'overall testing effectiveness

Using a BN tool such as [13] or [18] the various subnets are joined, according to the BN object approach [19–21] as shown in Fig. 10. Here each box represents a BN where only the 'input' and 'output' nodes are shown. For example, for the BN representing the defects in phase 2 the 'input' node residual defects pre is defined by the marginal distribution of the output node residual defects post of the BN representing the defects in phase 1.

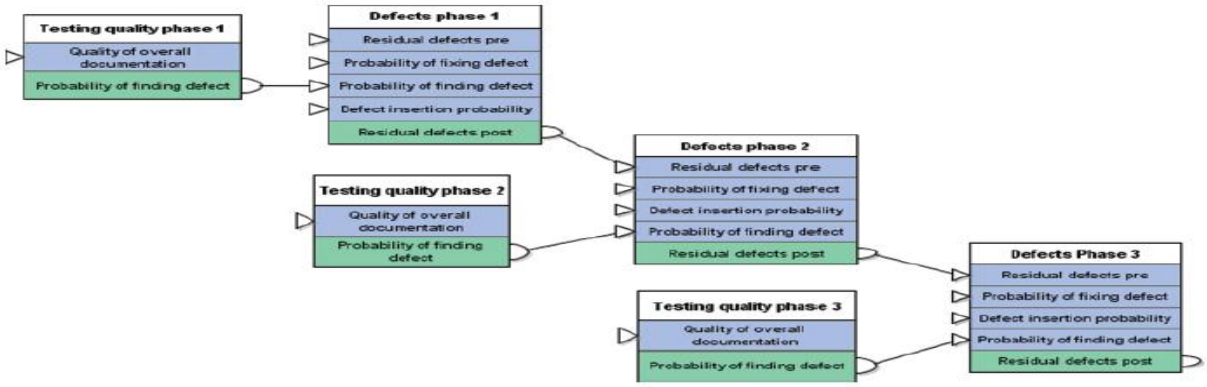

Fig. 10 Sequence of software testing phases as linked BN objects

The general structure of the BN model proposed here is relevant for any software development organization whose level of maturity includes defined testing phases in which defect and effort data is recorded. However, it is important to note that a number of the key probability distributions will inevitably be organization/project specific. In particular, there is no way of producing a generic distribution for the 'probability of finding a defect' in any given phase (and, as is discussed below this is especially true of the operational testing phase); indeed, even within a single organization this distribution will be conditioned on many factors (such as ones that are unique to a particular project) that may be beyond the scope of a workable BN model. At best it can be assumed that there is sufficient maturity and knowledge within an organization to produce a 'benchmark' distribution in a given phase. Where necessary this distribution can then still be tailored to take account of specific factors that are not incorporated in the BN model. It is extremely unlikely that such tailoring will always be able to take account of extensive relevant empirical data; hence, as in most practically usable BN models, there will be a dependence on subjective

judgments. However, at least the subjective judgments and assumptions are made explicit and visible. The assumptions about the 'probability of finding a defect' are especially acute in the case of the operational testing phase because, for example, in this phase the various levels of 'operational usage' will be much harder to standardize on. What is being done here is effectively the prediction of the reliability and to do this accurately may require the operational usage node to be conditioned on a formally defined operational profile such as described in the literature on statistical testing [22].

DYNAMIC DISCRETIZATION

While the results in the commercial validation studies referenced in section 3 were promising, the 'Achilles' heel' of using BNs for this type of application was soon revealed. The traditional approach to handling (non-Gaussian) continuous nodes is static: these nodes have to be discretized using some predefined range and intervals. This is cumbersome, error prone, and highly inaccurate because it assumes the analyst knows in advance which ranges will contain most of the probability mass for each node in the model under many different scenarios. As a very simple example, for one organization the size (measured in KLOC) was an independent variable used to derive empirical priors for defects inserted. Empirical data gave a prior for 'size' whose distribution had a mean of 15 KLOC, with most 'modules' being less than 50 KLOC. The resulting statically discretized model (taking account of the potentially large range of the 'continuous' nodes) for the particular testing phase is shown in Fig. 11.

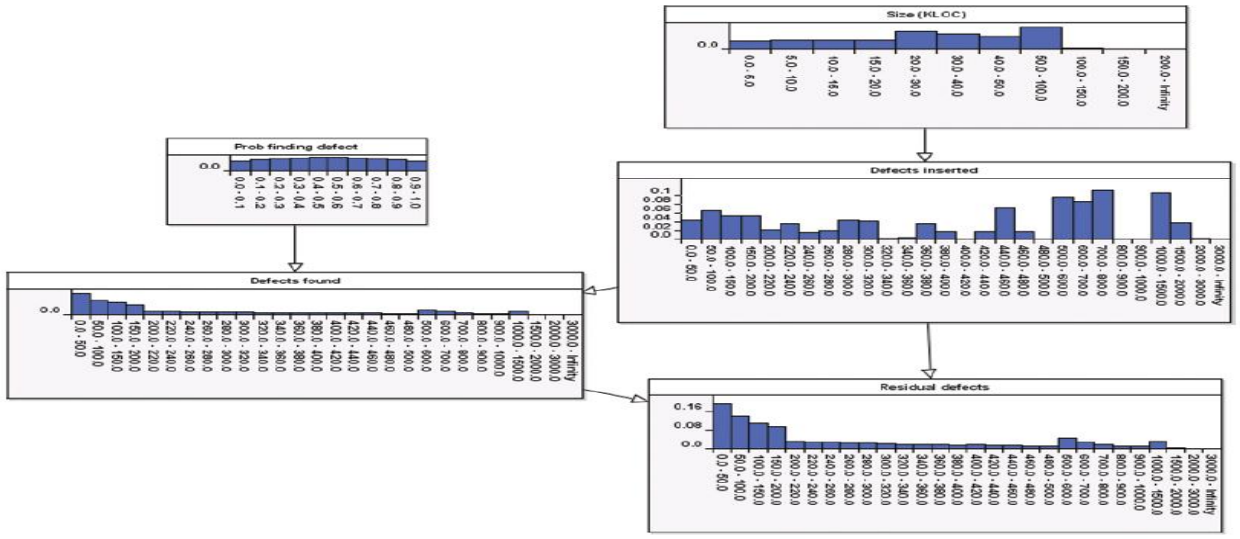

Fig. 11 Statically discretized defect model with marginal distributions

In the model the CPD for the node 'defects found' is defined as a binomial distribution with p being the 'probability of finding a defect' and n being the 'defects inserted'. The CPD for the node 'residual defects' is simply defined by the deterministic function 'defects found' minus 'defects inserted'. As with any attempt at discretization, there is a need to balance the number of states (accuracy) against computational speed. There was much discussion, agonizing and continual refinement of the discretizations. While predictions were generally good within the

'expected' range (i.e. less than 50 KLOC) there were wild inaccuracies for modules whose properties were not 'typical'. The inaccuracies were inevitably due to discretization 'errors'. For example, the model cannot distinguish between any modules whose size is in the range from 50 to 100 KLOC, so a module of size 51 KLOC is treated identically to one of 99 KLOC, while if say 1005 defects are found then the model cannot distinguish such an observation from 1499 defects being found. Such inaccuracies, as well as the wasted effort over selecting and defining discretization intervals, can now be avoided by using a technique called dynamic discretization that is described in [11] (being based on the original work in [23]) and implemented in [13]. This dynamic discretization algorithm, works for hybrid BNs (meaning that the nodes can be either discrete or continuous). Any node that is to be treated as continuous is simply flagged in the model and the modeller only has to specify a range (such as zero to one for the 'probability of finding a defect' node and zero to infinity for the 'size KLOC' node). The resulting dynamically discretized model is shown in Fig. 12.

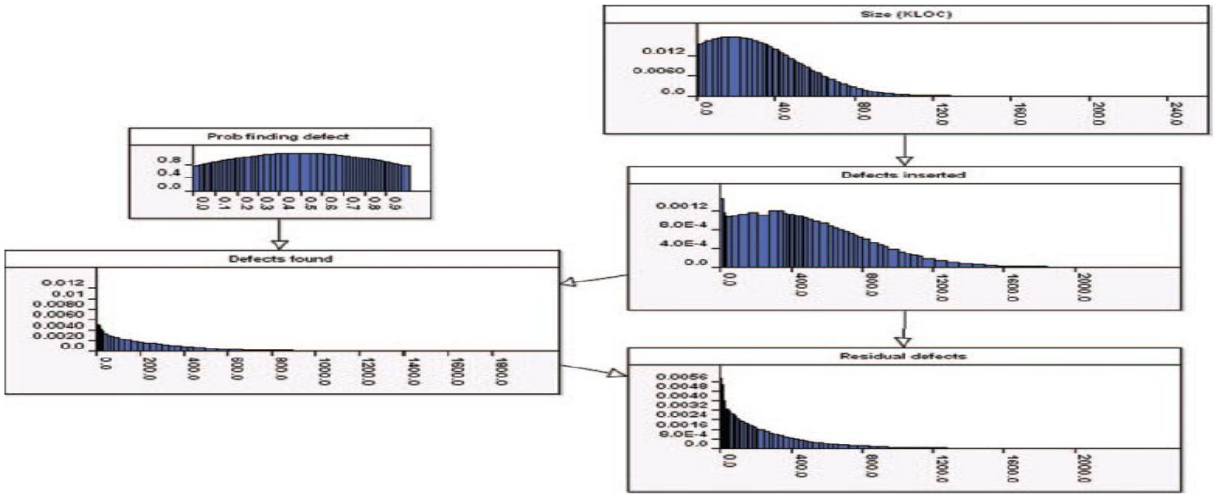

Fig. 12 Dynamically discretized defect model with marginal distributions

The dynamic discretization algorithm uses entropy error [23] as the basis for approximation. In outline, the algorithm follows these steps.

1. Convert the BN to a junction tree (JT) and choose an initial discretization for all continuous variables.
2. Calculate the node probability table (NPT) of each node given the current discretization.
3. Enter evidence and perform global propagation on the JT, using standard JT algorithms.
4. Query the BN to get posterior marginals for each node, compute the approximate relative entropy error, and check if it satisfies the convergence criteria.
5. If not, create a new discretization for the node by splitting those intervals with the highest entropy error.
6. Repeat the process by recalculating the NPTs and propagating the BN, and then querying to get the marginals and then split intervals with the highest entropy error.
7. Continue to iterate until the model converges to an acceptable level of accuracy.

This dynamic discretization approach allows more accuracy in the regions that matter and incurs less storage space over static discretizations. In the implementation [13] of the algorithm the user can select the number of iterations and convergence criteria, and hence can go for an arbitrarily high precision (at the expense of increased computation times). A detailed formal analysis of the computational complexity of the algorithm is the subject of ongoing research because it is so highly dependent on the particular BN topology and the location of the dynamically discretized nodes. BN propagation is known generally to be NP-hard [24] and this algorithm will, like any BN algorithm, be unable to cope with models in which the underlying graph clique sizes are large. However, as illustrated in the following the dynami discretization (DD) algorithm normally runs almost as efficiently as the non-DD algorithm when the number of iterations is set at around 20.

To compare the differences in accuracy between the DD versus non-DD version of the above model, Table 3 shows the results achieved (rounded to integer values) in a number of scenarios where it is possible to calculate the mean of the expected outcome analytically. For example, when the KLOC size is 100 and it is known that 250 defects are found, then given the prior assumptions about defects inserted the expected mean for the node 'residual defects' is 1250. The mean of this node predicted in the DD version is 1250, but in the non-DD version the mean is 825. The table clearly shows consistently accurate predictions for the DD version while the non-DD version is especially inaccurate in the more 'extreme'regions. For example, when the probability of finding a defect is 0.71 and 4000 defects are inserted than the mean of the residual defects is 1160; whereas the DD model gets this right the non-DD model predicts a mean of 14 853 primarily because 4000 defects lies within the end interval [3000 to infinity]. All of these
calculations were done with the number of iterations set to 20; the calculation time for a model of this size is almost indistinguishable from the non-DD case (less than 8 s on a standard PC).

Table 3 Comparison between DD and non-DD version under different scenarios

KLOC	—	—	—	—	100	5	50	250	200	10	50	0.5
Probability finding defect	0.38	0.71	0.71	—	—	—	—	—	0.5	0.5	0.5	0.5
Defects inserted	2000	100	4000	0	—	—	—	—	—	—	—	—
Defects found	—	—	—	—	250	100	100	100	—	—	—	—
Mean of predicted residual defects (non-DD)	1154	34	14 853	25	825	25	592	2429	3654	65	367	35
Mean of predicted residual defects (DD)	1240	29	1160	0	1250	3	649	3650	1500	75	375	4
Expected residual defects (analytic)	1240	29	1160	0	1250	0*	650	3650	1500	75	375	4

*The prior expected defects inserted in this scenario is 45, but 100 defects are subsequently found. The BN model therefore correctly infers that more than the number of defects inserted is a distribution in the range 100 to infinity.

A more comprehensive comparative analysis between two versions of a commercial-scale model (including a comparison on computation times) is described in [25]. Using the data of the projects described in [16] it was found that where predictive accuracy in the non-DD version was poor, much of this inaccuracy was due to the static discretization and significantly improved predictions were achieved in the DD version of the model; a typical example would be where the actual defects were outside the 'expected' range – an actual value of say 1700 was predicted

as 1000 with non-DD but 1500 with DD. For large-scale models at high accuracy settings computation time can be significantly longer than non-DD versions but since such models are rarely required to run in 'real-time' this is not normally a concern.

CONCLUSIONS

For organizations that already collect software metrics data there is a compelling argument for using BNs to predict software reliability and defects. The causal models enable simple explanatory factors, such as testing effort, to be incorporated which can have a major impact on the resulting predictions. While such BN models have proven to be useful, their accuracy was traditionally constrained by the following two factors.

1. The inevitable subjectivity resulting from expert elicitation (since there is never sufficient objective data alone on which to build such models).
2. The static discretization necessary in BN inference algorithms. While there is little that can be done to dramatically improve the first of these, it has been shown that the second is largely solvable. The radical new approach to inference using dynamic discretization removes the previous constraints and also makes it much easier to build and modify BN models with 'continuous' nodes. This has resulted in significantly more accurate predictions with only minimal increases in computation time.

REFERENCES

1. Fenton, N. E. and Neil, M. A critique of software defect prediction models. IEEE Trans. Softw. Engng, 1999, 25(5), 675–689.
2. Gras, J.-J. End-to-end defect modeling. IEEE Soft., 2004, 21(5), 98–100.
3. Hall, P., May, J., Nichol, D., Czachur, K., and Kinch, B. Integrity prediction during software development. In Proceedings of the IFAC Symposium on Safety of computer control systems 1992 (SAFECOMP'92), computer systems in safety-critical applications, Zurich, Switzerland, 1992, pp. 239–244 (Pergamon Press, Oxford).
4. Neil, M. and Fenton, N. E. Predicting software quality using Bayesian belief networks. In Proceedings of the 21st Annual software engineering workshop, NASA Goddard Space Flight Centre, 1996, pp. 217–230 (NASA, Maryland).
5. Ziv, H. and Richardson, D. J. Bayesian-network confirmation of software testing uncertainties. In Proceedings of the Sixth European Software Engineering Conference (ESEC), Zurich, 22–25 September 1997.
6. Dahll, G. Combining disparate sources of information in the safety assessment of software-based systems. Nucl. Engng Des., 2000, 195(3), 307–319.
7. Littlewood, B., Strigini, L., Wright,D., Fenton, N. E., and Neil, M. Bayesian belief networks for safety assessment of computer-based systems. In System performance evaluation methodologies and applications, (Ed. E. Gelenbe), 2000, pp. 349–364 (CRC Press, Boca Raton, FL).
8. Amasaki, S., Mizuno, O., Kikuno, T., and Takagi, Y. A Bayesian belief network for predicting residual faults in software products. In Proceedings of 14th International Symposium on Software reliability engineering (ISSRE2003), Denver, Colorado, November, 2003, pp. 215–22 (IEEE Computer Society Press).

9. Bibi, S. and Stamelos, I. Software process modeling with Bayesian belief networks. Tenth International SoftwareMetrics Symposium (Metrics 2004), Chicago, IL, USA, 2004.
10. Fenton, N. E., Neil, M., and Gallan, J. Using ranked nodes to model qualitative judgements in Bayesian networks. IEEE Trans. Knowl. Data Engng, 2007, 19(10), 1420–1432.
11. Neil, M., Tailor, M., and Marquez, D. Inference in hybrid Bayesian networks using dynamic discretization. Stat. Comput., 2007, 17(3), 219–233.
12. Fenton, N. E. and Ohlsson, N. Quantitative analysis of faults and failures in a complex software system. IEEE Trans. Softw. Engng, 2000, 26(8), 797–814.
13. Agena Ltd. AgenaRisk, 2007, available from http://www. agenarisk.com. JRR161 _ IMechE 2008 Proc. IMechE Vol. 222 Part O: J. Risk and Reliability
14. Pe´rez-Min˜ana, E. and Gras, J.-J. Improving fault prediction using Bayesian networks for the development of embedded software applications: research articles. Softw. Test. Verif. Reliab., 2006, 16(3), 157–174.
15. Wang, H., Peng, F., Zhang, C., and Pietschker, A. Software project level estimation model framework based on Bayesian belief networks. In Proceedings of the Sixth International Conference on Quality software (QSIC'06), Beijing, China, 27–28 October, 2006, pp. 209–218 (IEEE Computer Society Press).
16. Fenton, N. E., Neil, M., Marsh, W., Hearty, P., Marquez, D., Krause, P., and Mishra, R. Predicting software defects in varying development lifecycles using Bayesian nets. Inform. Softw. Technol., 2007, 49, 32–43.
17. Fenton, N. E., Neil, M., and Krause, P. Software measurement: uncertainty and causal modelling. IEEE Softw., 2002, 10(4), 116–122.
18. Hugin A/S. Hugin expert, 2007, available from http://www.hugin.com.
19. Bangsø, O. and Wuillemin, P. H. Top-down construction and repetitive structures representation in Bayesian networks. In Proceedings of 13th International Florida artificial intelligence research symposium, FL, USA, 2000, pp. 282–286 (AAAI Press, Menloe Park, CA).
20. Koller, D. and Pfeffer, A. Object-oriented Bayesian networks. In Proceedings of the 13th Annual Conference on Uncertainty in AI (UAI), Providence, RI, 1997, pp. 302–313 (Morgan Kaufmann, San Francisco).
21. Neil, M., Fenton, N., and Nielsen, L. Building largescale Bayesian networks. Knowl. Engng Rev., 2000, 15(3), 257–284.
22. Dyer, M. The cleanroom approach to quality software development, 1992 (John Wiley & Sons, New York).
23. Kozlov, A. V. and Koller, D. Nonuniform dynamic discretization in hybrid networks. In Proceedings of the 13th Annual Conference on Uncertainty in AI (UAI), Providence, RI, 1997, pp. 314–325 (AAAI Press, Menloe Park, CA).
24. Cooper, G. F. The computational complexity of probabilistic inference using Bayesian belief networks. Artif. Intell., 1990, 42(2–3), 393–405.
25. Fenton, N. E., Radlinski, L., and Neil, M. Improved Bayesian networks for software project risk assessment using dynamic discretisation. In Software engineering techniques: design for quality (Proceedings of software engineering techniques 2006, Warsaw, Poland, 17–20 Oct 2006), (Ed. K. Sacha), 2006, pp. 139–148 (Springer, Boston, MA).