Using Bayesian Networks to Predict Software Defects and Reliability

Norman Fenton
Dep. of Computer Science
Queen Mary, University of London,
and Agena Ltd, London
norman@dcs.qmul.ac.uk

Martin Neil
Dep. of Computer Science
Queen Mary, University of London,
and Agena Ltd, London
martin@dcs.qmul.ac.uk

David Marquez
Dep. of Computer Science
Queen Mary, University of London
marquezd@dcs.qmul.ac.uk

Abstract

This paper reviews the use of Bayesian Networks (BNs) in predicting software defects and software reliability. The approach allows us 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 by organisations such as Motorola, Siemens and Philips who have reported accurate predictions. 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 modellers to have to pre-define discretisation intervals in advance and resulted in very inaccurate predictions where the range, for example, of defect counts was large. Fortunately, recent advances in BN algorithms now enable us to use so-called dynamic discretisation for any continuous nodes. This results in significantly improved accuracy for defects and reliability prediction type models.

1. Introduction

At an international software metrics conference some years ago a well known and highly respected metrics guru 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 didn't. To do this they 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 characterised such successful projects. Over a couple of years a number of such 'star' projects were identified, including some which achieved the magical perfect reliability target of zero defects per KLOC in the first six months post-release. But, it turned out that what they learned from this was very different to what they had expected. Almost every one of the star projects was, in fact, an unmitigated disaster. The reason for the very low number of defects reported by customers was that they were so bad that they never got used.

Therein lies the classic weakness of traditional software metrics that has been highlighted in [Fenton and Neil 1999] and [Gras 2004] – the omission of sometimes obvious and simple causal factors that can have a major explanatory effect on what is observed and learnt. If you are managing a software development project and you hear 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 you heard that a large number of defects were discovered. It seems quite obvious that 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 such causal factors. But until relatively recently such an obvious approach was largely ignored. This paper reviews how the problem has been addressed by Bayesian Networks (BNs). We explain the basis of the solution by means of a very simple BN in Section 2, while in Section 3 we show the kind of commercial-scale BN models that have been implemented by organisations like Motorola, Siemens and Philips. In Section 4 we focus on how a major weakness of the BN models has now been solved. Specifically, we summarise how recent work on dynamic discretisation algorithms has largely fixed the “Achilles’ Heel” of handling continuous nodes properly. A companion paper [Marquez et al 2007] describes in detail how the new dynamic discretisation algorithm can be used to address a range of reliability problems.

2. A simple causal model for software defect prediction

The case for using BNs as causal models for software defects and reliability prediction is simple and compelling.

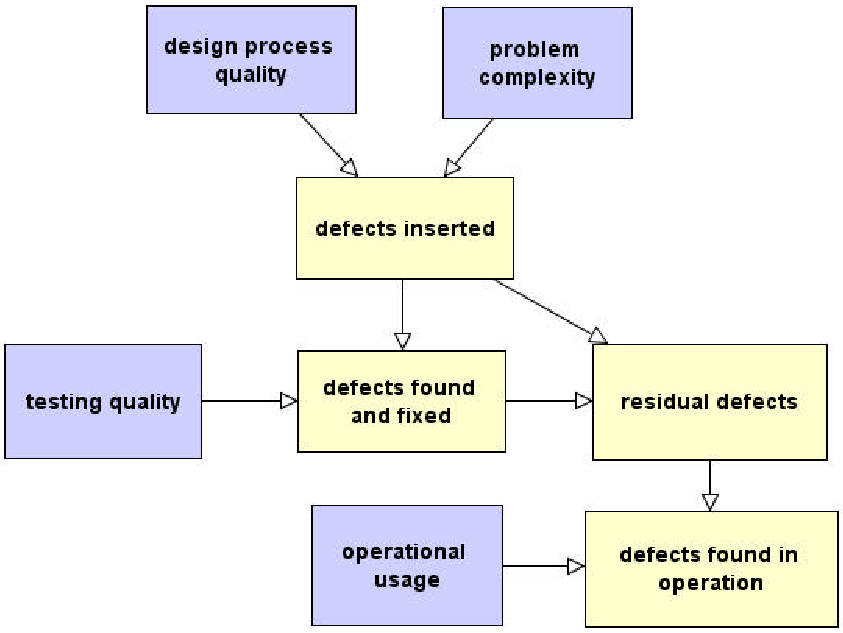


Figure 1: Simplified version of a BN model for software defects and reliability prediction

Figure 1 shows a simple example of such a model. Here the number of operational defects (those found by customers) in a software module is what we are really interested in predicting. We know this is clearly dependent on the number of residual defects. But it is also critically dependent on the amount of operational usage. If you do not use the system you will find no defects irrespective of the number there (the model allows you to predict this with perfect precision). The number of residual defects is determined by the number you introduce during development minus the number you successfully find and fix. Obviously defects found and fixed is dependent on the number introduced. The number introduced is influenced by problem complexity and design process quality. The better the design the fewer the defects and the less complex the problem the fewer defects. Finally, how many defects you find is influenced not just by the number there to find but also by the amount of testing effort.

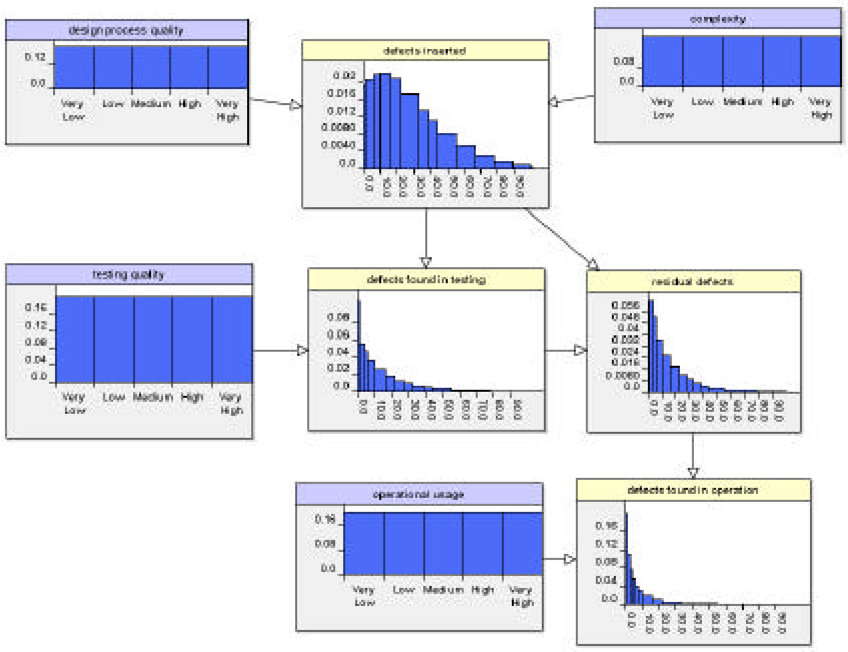


Figure 2: BN Model with marginal distributions for variables superimposed on nodes

Figure 2 shows the marginal distributions — this represents our uncertainty before we enter any specific information about this module. What it means, for example, is that the module is just as likely to have very high complexity as very low, and that the number of defects found and fixed in testing is in a wide range where the median value is about 18-20 (the prior distributions here were for a particular organisation's modules). As we enter observations about the module the probability distributions update as shown in Figure 3.

Here we have entered the observation that this module had zero defects found and fixed in testing and that the problem complexity is 'High'. Note that all the other distributions changed. The model is doing both forward inference to predict defects in operation and backwards inference about, say, design process quality. Although the fewer than expected defects found does indeed lead to a belief that the post-release faults will drop, the model shows that most likely explanation is inadequate testing.

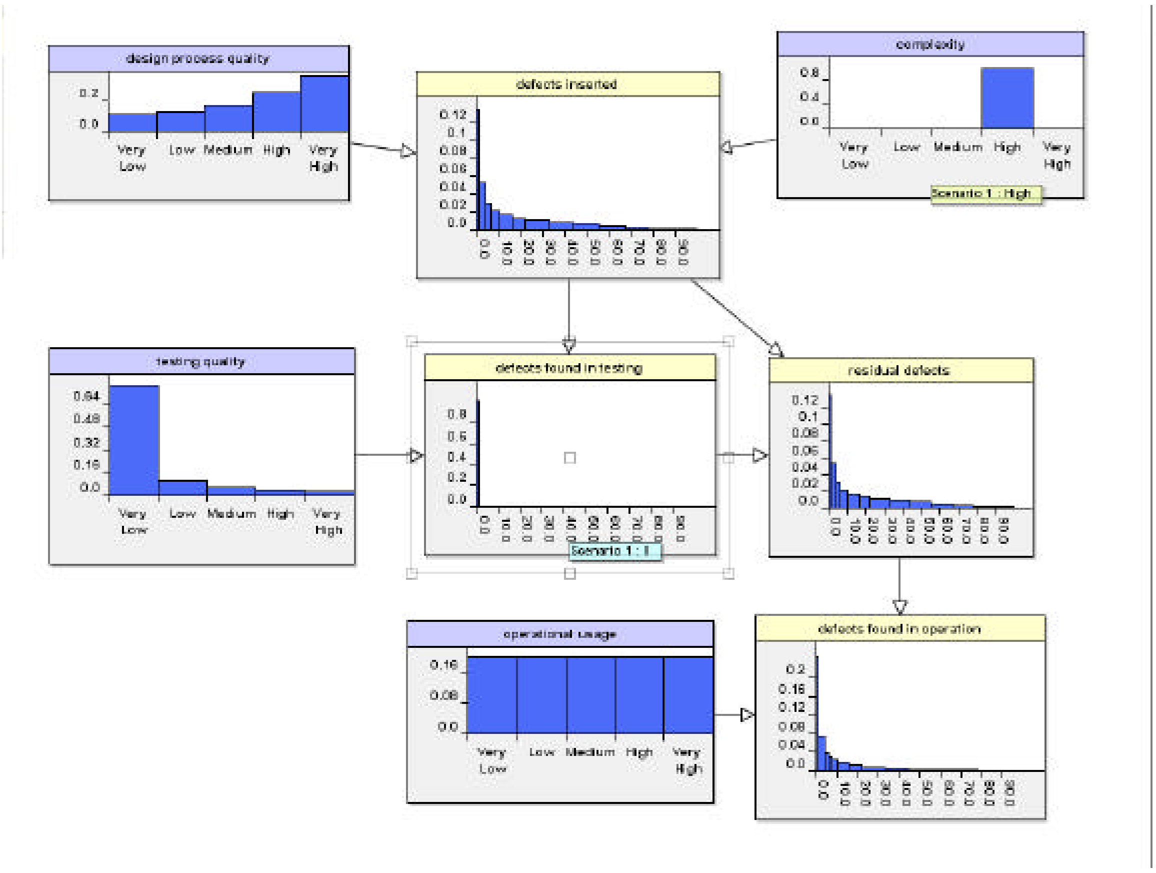


Figure 3: Zero defects in testing and high complexity observed

So far we have made no observation about operational usage. If, in fact the operational usage is “Very High” (Figure 4) then what we have done is replicate the apparently counter-intuitive empirical observations [Fenton and Neil 1999] whereby a module with no defects found in testing has a high number of defects post-release.

But suppose we find out that the test quality was “Very High” (Figure 5)

Then we completely revise out beliefs. We are now pretty convinced that the module will be fault free in operation. Note also that the ‘explanation’ is that the design process is likely to be very high quality.

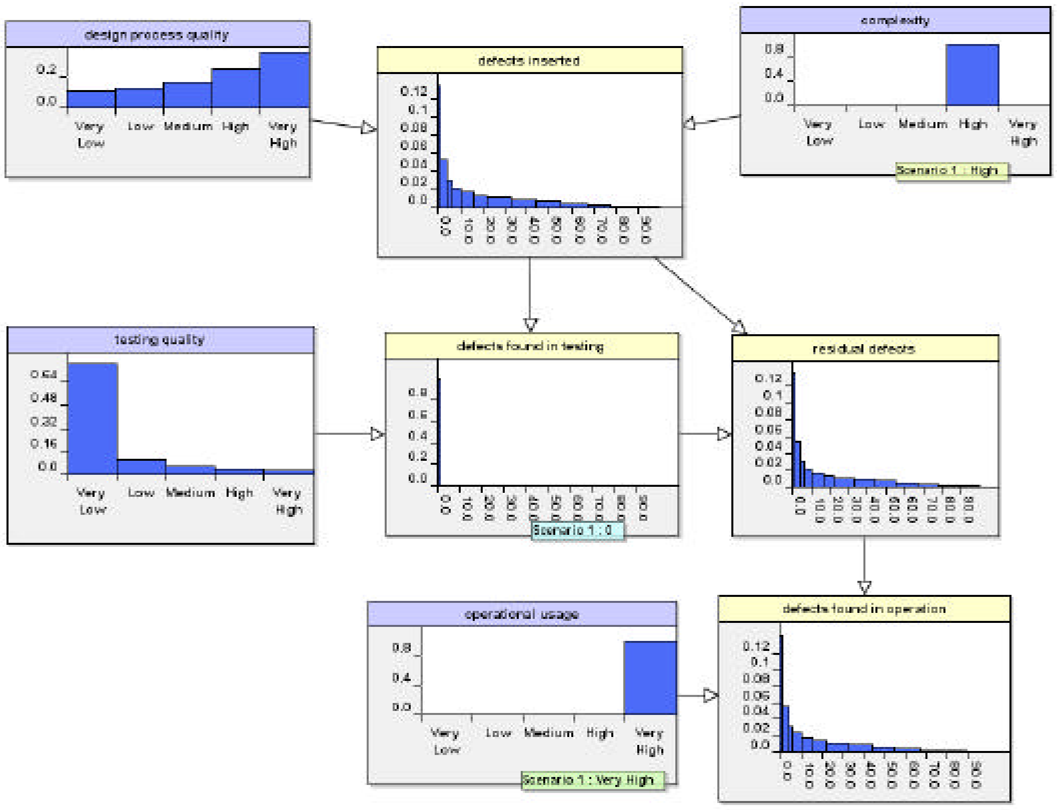


Figure 4: Very high operational usage

3. Commercial scale versions of the defect prediction models

The ability to do the kind of prediction and what-if analysis described in Section 2 has proved to be very attractive to organisations who need to monitor and predict software defects and reliability, and who already collect defect-type metrics. Hence, organisation such as Motorola [Gras 2004] [Minana and Gras 2006], Siemens [Wang et al 2006] and Philips [Fenton et al 2007] have exploited models and tools originally developed in [Fenton et al 2002] 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 a sequence of testing phases (culminating in an operational testing phase to monitor and predict reliability) is used where each phase consists of a "subnet" like that in Figures 6, 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 defects fixing and insertion.

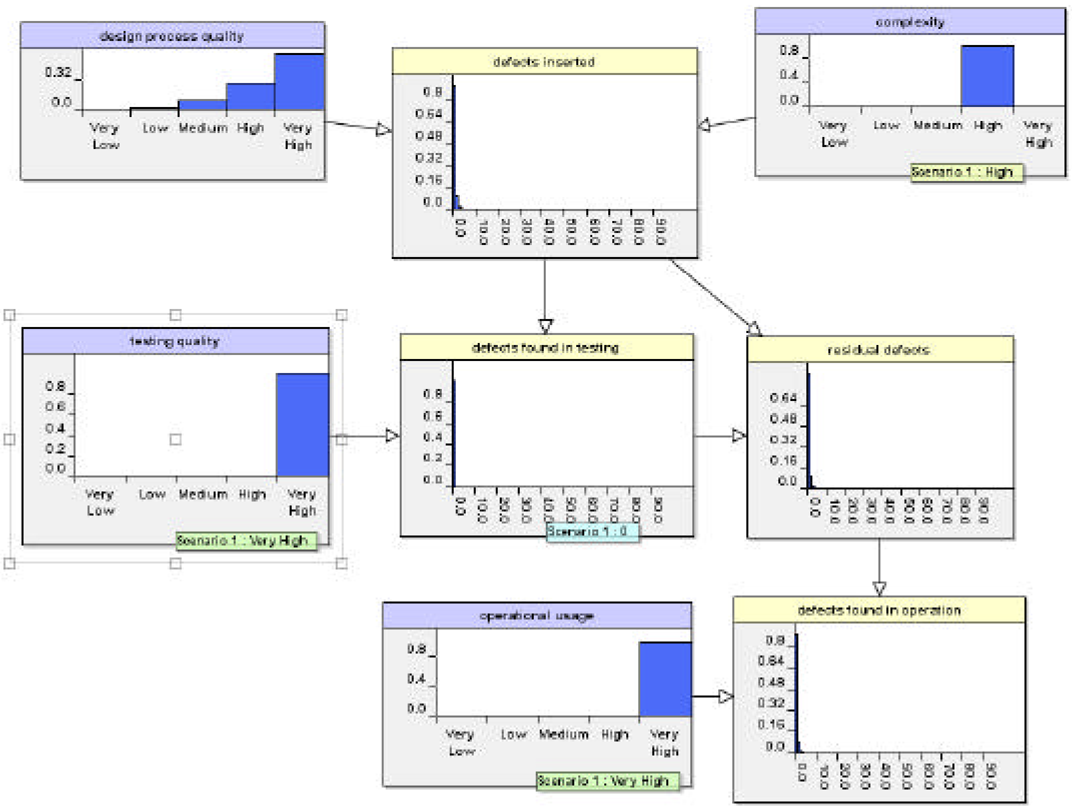


Figure 5: Testing quality very high

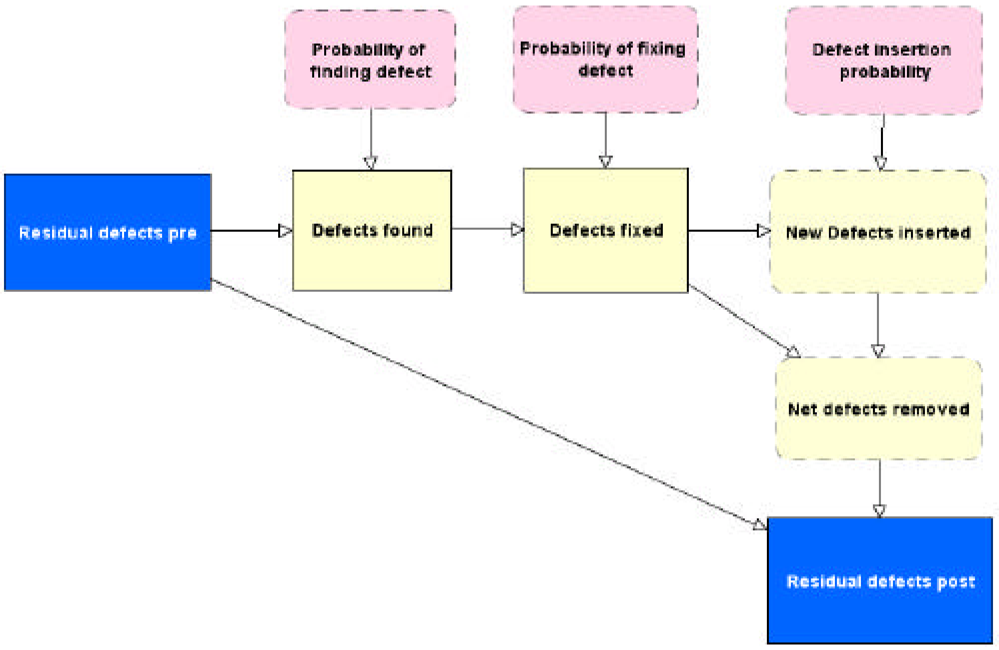


Figure 6: Defects phase subnet.

The distributions for nodes such as "probability of finding defect" derive from other subnets such as that shown in Figure 7. The particular nodes and distributions will, of course, vary according to the type of testing phase. Typically, phases such as: *unit*, *integration*, *system*, and *acceptance* testing were used pre-release.

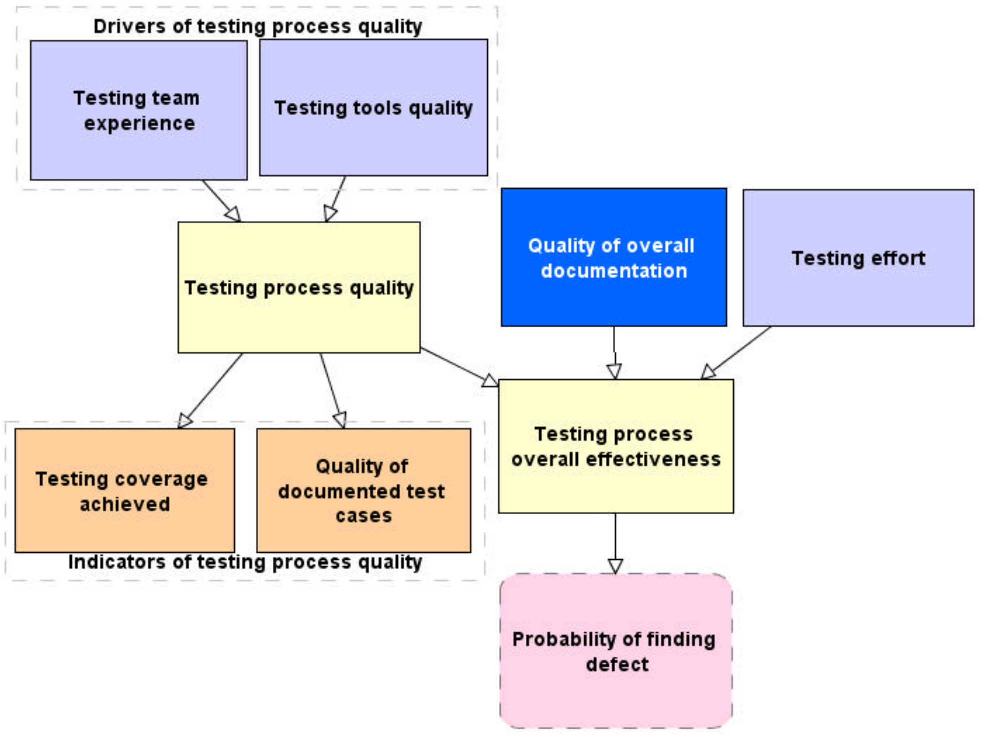
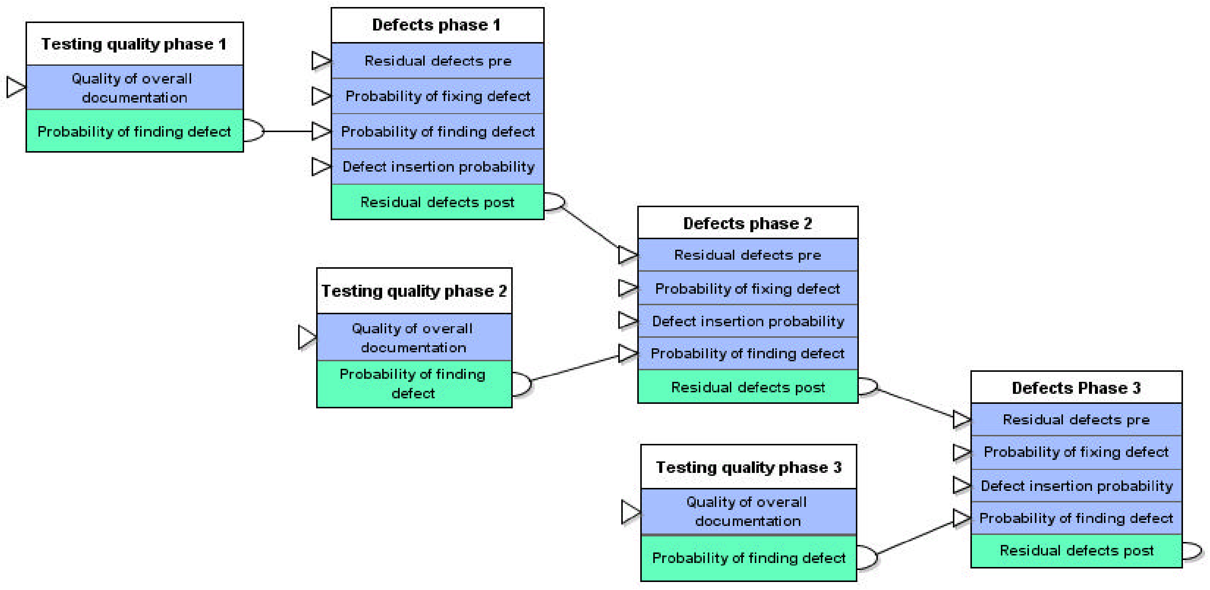

Drivers of testing process quality
Testing team experience
Testing tools quality
Testing process quality
Quality of overall documentation
Testing effort
Testing process overall effectiveness
Testing coverage achieved
Quality of documented test cases
Indicators of testing process quality
Probability of finding defect

Figure 7: Typical subnet for testing quality

Using a tool such as [Agena 2007] the various subnets are joined, according to the BN Object approach [Bangsø and Wuillemin 2000], [Koller and Pfeffer 1997], [Neil et al 2000] as shown in Figure 8. Here each box represents a BN where only the 'input' and 'output' nodes are shows. 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.



Testing quality phase 1
Quality of overall documentation
Probability of finding defect
Defects phase 1
Residual defects pre
Probability of fixing defect
Probability of finding defect
Defect insertion probability
Residual defects post
Testing quality phase 2
Quality of overall documentation
Probability of finding defect
Defects phase 2
Residual defects pre
Probability of fixing defect
Defect insertion probability
Probability of finding defect
Residual defects post
Testing quality phase 3
Quality of overall documentation
Probability of finding defect
Defects Phase 3
Residual defects pre
Probability of fixing defect
Defect insertion probability
Probability of finding defect
Residual defects post

Figure 8: Sequence of software testing phases as linked BN objects

4. The need for dynamic discretisation

While the results in the commercial validation studies referenced in Section 3 were very 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: you have to discretise such nodes using some pre-defined range and intervals. But this assumes the analyst can identify and appropriately discretise the high-density regions for each node in the model, and do so in advance of any inference taking place. This is cumbersome, error prone and highly inaccurate. As a very simple example, for one organisation the *size* (measured in KLOC, thousands of lines of code) was an independent variable used to derive empirical priors for defects inserted (empirical data gave a prior whose distribution had a mean of 15 times KLOC). Typically, the size of a 'module' was between 10 and 20 KLOC, but this was by no means consistent. The resulting statically discretised model (taking account of the potentially large range of the 'continuous' nodes) for the particular testing phase is shown in Figure 9. In the model the conditional probability distribution (CPD) for the node "defects found" is defined as a Binomial distribution with p= 'probability of finding a defect' and n = 'defects inserted'. The CPD for the node 'Residual defects' is the simply defined by the deterministic function 'defects found' minus 'defects inserted'.

As with any attempt at discretisation, there was a need to balance the number of states (accuracy) against computational speed. There was much discussion, agonising and continual refinement of the discretisations. While predictions were generally good within the 'expected' range there were wild inaccuracies for modules whose properties were not 'typical'. The inaccuracies were inevitably dues to discretisation 'errors'. For example, the model cannot distinguish between any modules whose size is in the range 50 to 100 KLOC, so a module of size 51 KLOC is treated identically to one of 99KLOC, while if we observe say 1005 defect found then the model cannot distinguish such an observation from 1999 defects found.

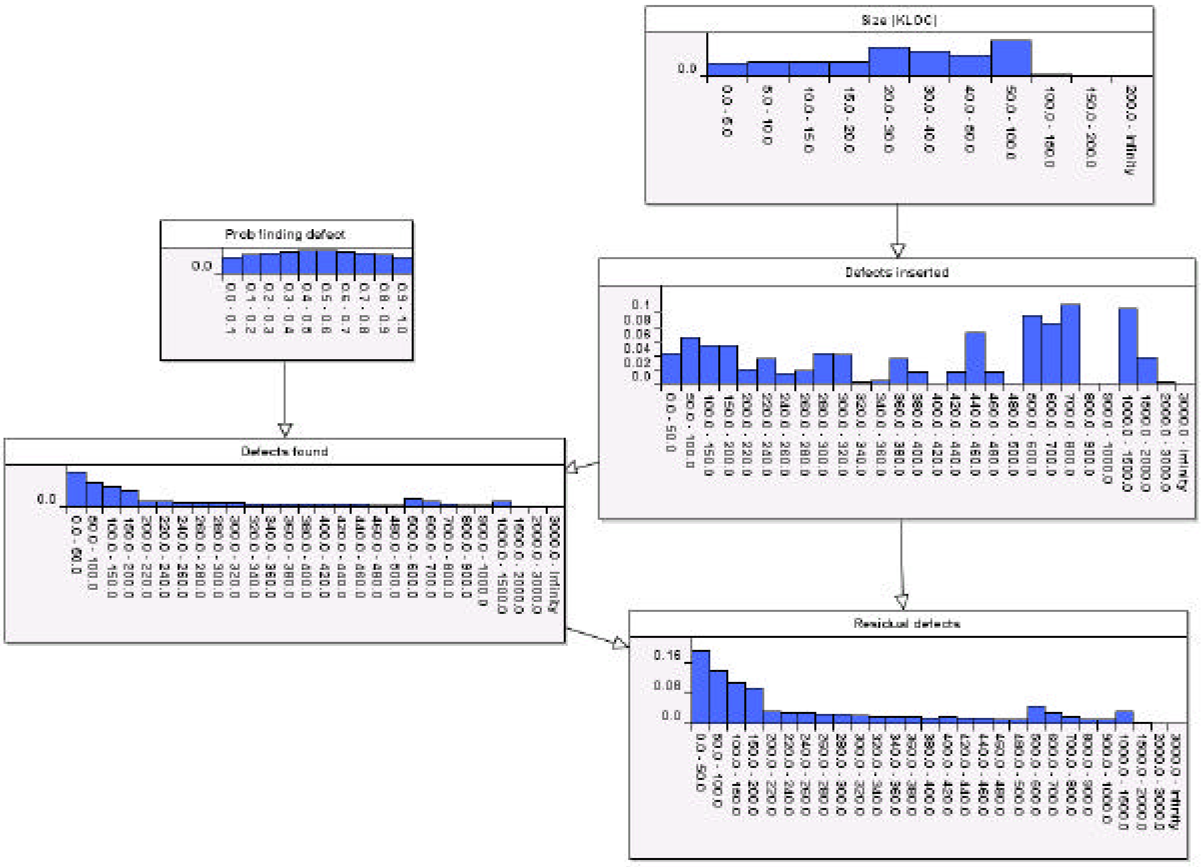


Figure 9: Statically dscretised defects model with marginal distributions

Such inaccuracies, as well as the wasted effort over selecting and defining discretisation intervals, are now completely avoided by using the new dynamic discretisation described in [Neil et al 2007] and implemented in [Agena 2007]. This dynamic discretisation 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 0 to 1 for the ‘probability of finding a defect’ node and zero to infinity for the ‘size KLOC’ node). The resulting dynamically discretised model is shown in Figure 10.

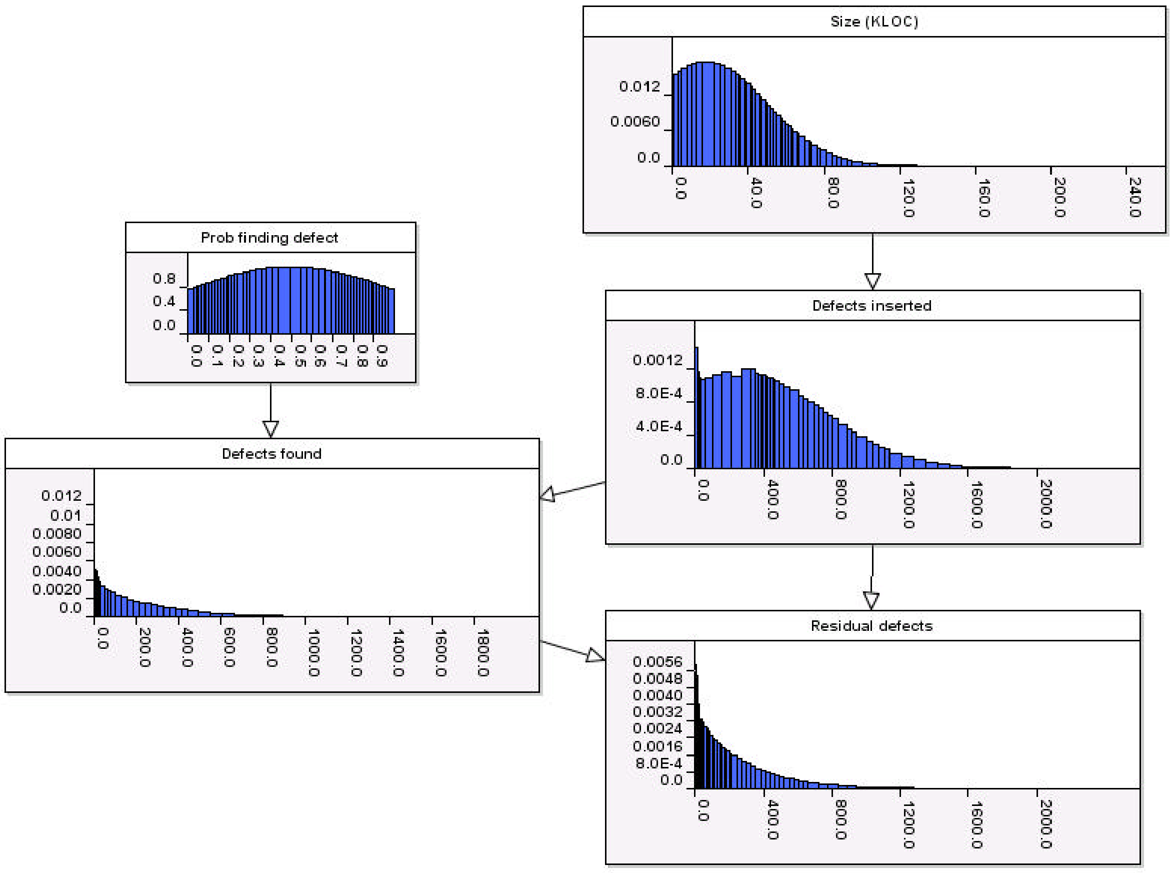


Figure 10: Dynamically discretised defects model with marginal distributions

The dynamic discretisation algorithm uses entropy error [Kozlov and Koller 1997] 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 discretisation for all continuous variables.
2. Calculate the Node Probability Table (NPT) of each node given the current discretisation.
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 discretisation for the node by splitting those intervals with 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 highest entropy error.
7. Continue to iterate until the model converges to an acceptable level of accuracy.

This dynamic discretisation approach allows more accuracy in the regions that matter and incurs less storage space over static discretisations. In the [Agena2007] implementation of the algorithm the user can select the number of iterations and convergence criteria, and hence can go for arbitrarily high precision (at the expense of increased computation times).

To compare the differences between the DD versus non-DD version of the above model Table 1 shows the results achieved (rounded to zero decimal places) 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 14853 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 seconds on a standard PC).

Table 1

KLOC	—	—	—	—	100	5	50	250	200	10	50	0.5
Prob. 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	14853	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 number of defects inserted is a distribution in he 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 [Fenton et al 2006]. 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.

5. Conclusions

For organisations that already collect software metrics data there is a compelling argument for using BNs to predict software defects and reliability. The causal models enable us to incorporate simple explanatory factors, such as testing effort, 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 static discretisation necessary in BN inference algorithms. The radical new approach to inference using dynamic discretisation removes the previous constraints and also makes it much easier to build and modify BN models with 'continuous' nodes. Extremely accurate predictions are therefore now possible with only minimal increases in computation time.

Acknowledgements

The authors acknowledge the contribution of Peter Hearty, Paul Krause, Rajat Mishra, Lukasz Radlinksi, and Manesh Tailor to this work, the support of EPSRC and Agena in the projects, eXdecide (EP/C005406/1) and DyFusion, and the EU in the project MODIST (IST-2000-28749).

References

Agena (2007) *AgenaRisk: Advanced risk analysis for important decisions*. http://www.agenarisk.com

O. Bangsø, P. H. Wuillemin (2000), Top-down construction and repetitive structures representation in Bayesian networks, Proceedings of The Thirteenth International Florida Artificial Intelligence Research Symposium Conference, Florida, USA., pp. 282-286.

S. Bibi, I. Stamelos, Software Process Modeling with Bayesian Belief Networks, Proceedings of 10th International Software Metrics Symposium (Metrics 2004) 14-16 September 2004, Chicago, USA.

Fan, Chin-Feng, Yu, Yuan-Chang, BBN-based software project risk management, J Systems Software, 73 (2004) 193-203.

Fenton NE, Neil M (1999), A Critique of Software Defect Prediction Models, *IEEE Transactions on Software Engineering*, 25(5) (1999) pp. 675-689 .

Fenton NE, Krause P, Neil M (2002), Software Measurement: Uncertainty and Causal Modelling, *IEEE Software* 10(4) pp. 116-122.

Fenton NE, Neil M, Hearty P, Marsh W, Marquez D, Krause P, Mishra R (2007), Predicting Software Defects in Varying Development Lifecycles using Bayesian Nets, *Information & Software Technology*, Vol 49, pp 32-43.

Fenton NE, Radlinski L, Neil M (2006), Improved Bayesian Networks for Software Project Risk Assessment Using Dynamic Discretisation, *Software Engineering Techniques (SET 2006),* Warsaw, Poland, 17-20.

Gras JJ (2004) End to end defect modelling, *IEEE Software* 21(5), pp 98-100

Minana EP, and Gras JJ (2006), Improving fault prediction using Bayesian networks for the development of embedded software applications, *Software Testing, Verification and Reliability* 16(3), pp. 157-174

Koller D, Pfeffer A (1997), Obje ct-Oriented Bayesian Networks, Proceedings of the Thirteenth *Conference on Uncertainty in Artificial Intelligence (UAI97),* Providence, Rhode Island, USA, , pp. 302-313.

Kozlov A.V, Koller D. (1997). Nonuniform dynamic discretization in hybrid networks, in D. Geiger and P.P. Shenoy (eds.), *Uncertainty in Artificial Intelligence*, 13, pp. 314-325

Neil, M Fenton NE, Nielsen L (2000), Building large-scale Bayesian Networks, *The Knowledge Engineering Review*, 15(3), pp. 257-284.

Neil M., Tailor M. and Marquez D. (2007) Inference in Bayesian Networks using dynamic discretisation, to appear *the Journal of Statistics and Computing*.

Marquez D, Neil M., Fenton NE (2007), A new Bayesian Network approach to Reliability modelling. *Mathematical Methods in Reliability (MMR07*), Glasgow

Wang H, Peng F, Zhang C, Pietschker A (2006), Software Project Level Estimation Model Framework based on Bayesian Belief Networks, *Sixth International Conference on Quality Software (QSIC'06*) pp. 209-218.