Proceedings of 21st Annual Software Engineering Workshop
NASA/Goddard Space Flight Centre, December 4-5, 1996

Predicting Software Quality using Bayesian Belief Networks

Martin Neil & Norman Fenton
Centre for Software Reliability
City University
Northampton Square
London EC1V OHB, UK

Abstract

In the absence of an agreed measure of software quality the *density of defects* has been a very commonly used surrogate measure. As a result there have been numerous attempts to build models for predicting the number of residual software defects. Typically, the key variables in these models are either size and complexity metrics or measures arising from testing information. There are, however, serious statistical and theoretical difficulties with these approaches. Using Bayesian Belief Networks we can overcome some of the more serious problems by taking account of all the diverse factors implicit in defect prevention, detection and complexity.

1. Background

For the last 20 years the software engineering community has spent much effort in trying to answer the question, "Can we predict the quality of our software *before* we use it?". There are literally scores of papers, articles and reports advocating statistical models, metrics and solutions which purport to answer this question. Generally, efforts have tended to concentrate solely on one of the following three problem perspectives:

a) Predicting the number of defects in the system using software size and complexity metrics

The earliest study of the relationship between defects and complexity appears to have been [Akiyama,1971] which was based on a system developed at Fujitsu, Japan. It is typical of many regression based "data fitting" models which became common-place in the literature (such as [Ferdinand 1974], [Lipow 1982], [Gaffney 1984], [Basili and Perricone 1984], [Shen 1985], [Compton and Withrow 1990], [Moller and Paulish 1993]). The study showed that linear models of some simple metrics provide reasonable estimates for the total number of defects *d* (the dependent variable) which is defined as the sum of the defects found during testing and the defects found during two months after release. Although there is no convincing evidence to show that any of the hundreds of published complexity metrics are good predictors of defect density, there *is* a growing body of evidence that some of these metrics may be useful in outlier analysis (especially when grouped together) [Bache and Bazzana 1993]—they can be used to predict which of a set of modules is likely to be especially defect-prone.

b) Inferring the number of defects from testing information

Some of the most promising local models for predicting residual defects involve very careful collection of data about defects discovered during early inspection and testing phases. A notable example of this is reported by the IBM NASA Space shuttle team [Keller 1992]. Another class of testing metrics that appears to be quite promising for predicting defects is the class of so called *test coverage measures*. [Fenton and Pfleeger 1996]. For a given strategy and a given set of test cases we can ask what proportion of coverage has been achieved. The resulting metric is defined as the Test Effectiveness Ratio (TER) with respect to that strategy. Clearly we would expect defect rate to decrease as the values of these metrics increases. [Veevers and Marshall 1994] report on some defect and reliability prediction models using these metrics which give quite promising results.

c) Assessing the impact of design or process maturity on defect counts.

There are many experts who argue that the quality of the development process is the best predictor of product quality. The simplest metric of process quality is the 5-level ordinal scale SEI Capability Maturity Model ranking. Despite its widespread popularity, there is no convincing evidence to show that higher maturity companies generally deliver products with lower residual defect rate than lower maturity companies. Nevertheless, this seems to be a widely held assumption and is therefore important in explaining and predicting defects.

2. The need to take account of diverse factors

Despite the many efforts described above there appears to have been little overall improvement in the accuracy of the predictions made using these models (if predictions are *formally* made at all) or indeed whether the models make sense. Broadly speaking there are a number of serious statistical and theoretical difficulties that have caused these software quality prediction problems ([Neil 1992] provides explicit criticisms of many of the models). To avoid these problems we need to take account of all the diverse factors implicit in defect prevention, detection and complexity.

Perhaps the most critical issue in any scientific endeavour is agreement on the constituent elements or variables of the problem under study. Models are developed to represent the salient features of the problem in a systemic fashion. This is as much the case in physical sciences as social sciences. For instance, in macro-economic prediction we could not predict the behaviour of an economy without an integrated, complex, model of all of the known, pertinent variables. Choosing to ignore or forgetting to include key variables such as *savings rate* or *productivity* would make the whole exercise invalid and meaningless. Yet this is the position that many software practitioners are in - they are being asked to accept simplistic models which are missing key variables that are already *known* to be enormously important. Predicting the number of defects discovered based on lines of code alone is as much use as predicting a person's IQ from a knowledge of their shoe size.

Our view is that the isolated pursuit of these single issue perspectives on the quality problem are, in the longer-term, fruitless. The solution to many of the difficulties presented above is to develop prediction models that *unify* the diverse software quality prediction models. This unification will help produce new systematic models that better represent the complex relationships inherent in software engineering.

Only when such unified models are developed will statistical experimentation and then practical use be warranted.

As well as facing up to the complexity inherent in software engineering we must also recognise that modelling the actions of the designer and manager are crucial if we are to predict the quality of the final product. Again and again experience dictates that it is good managers and designers that determine the difference between failure and success. However researchers have tended to ignore the issue of human intervention even though we know it is *the* key variable in software design. A consequence of this is that subjectivity and uncertainty is all pervasive in software development. Project managers make decisions about quality and cost using best guesses; it seems to us that will always be the case and the best that researchers can do is a) recognise the fact and b) improve the 'guessing' process.

The results of inaccurate modelling and inference is perhaps most evident in the debate that surrounds the 'Is Bigger Better?' dilemma. This is the phenomenon that larger modules have lower defect densities [Basili and Perricone 1984] and [Shen 1985]. [Moller and Paulish 1993] provide further evidence, and also examined the effect of modifications and reuse on defect density. Similar experiences are reported by [Hatton 1993, 1994]. Basili and Perricone argued that this may be explained by the fact that there are a large number of interface defects distributed evenly across modules, and that larger modules tend to be developed more carefully. Others have mentioned the possible effects of testing.

The notion that larger modules have lower defect density is surprising because it questions the whole edifice of problem and design decomposition so central to software engineering. It suggests that building bigger modules will result in less defects overall. To act on these results would mean throwing away much of what is being advocated in structured, object-oriented and formal design - 'Why should we apply decomposition when it doesn't improve quality?'. Post-hoc explanations cannot easily dismiss the uncomfortable significance of this result.

3. Bayesian Belief Networks (BBNs)

Achieving the above modelling challenges appear onerous when one considers the tools previously available to researchers and practitioners. They have had to rely on the power of classical statistical analysis tools, such as regression, discriminant analysis and correlation. Classical methods demand simple linear structures and a wealth of data so often missing in software engineering. These methods have severely restricted the scale of problems that could be tackled. However, a relatively new but rapidly emerging technology has provided an elegant solution enabling us to push back the boundary of the problems that can be attacked: *Bayesian Belief Networks* (BBNs) [Pearl, 1988].

A BBN is a graphical network that represents probabilistic relationships among variables. BBNs enable reasoning under uncertainty and combine the advantages of an intuitive visual representation with a sound mathematical basis in Bayesian probability. With BBNs, it is possible to articulate expert beliefs about the dependencies between different variables and to propagate consistently the impact of evidence on the probabilities of uncertain outcomes, such as 'future system reliability'. BBNs allow an injection of scientific rigour when the probability distributions associated with individual nodes are simply 'expert opinions'. A BBN will derive all the implications of the beliefs that are input to it; some of these will be

facts that can be checked against the project observations, or simply against the experience of the decision makers themselves. There are many advantages of using BBNs, the most important being the ability to represent and manipulate complex models that might never be implemented using conventional methods. Because BBNs have a rigorous, mathematical meaning there are software tools that can interpret them and perform the complex calculations needed in their use. The specific tool used here is *Hugin Explorer* [Hugin 1996], which provides a graphical front end for inputting the BBNs in addition to a computational engine for the Bayesian analysis.

4. The Defect Density BBN

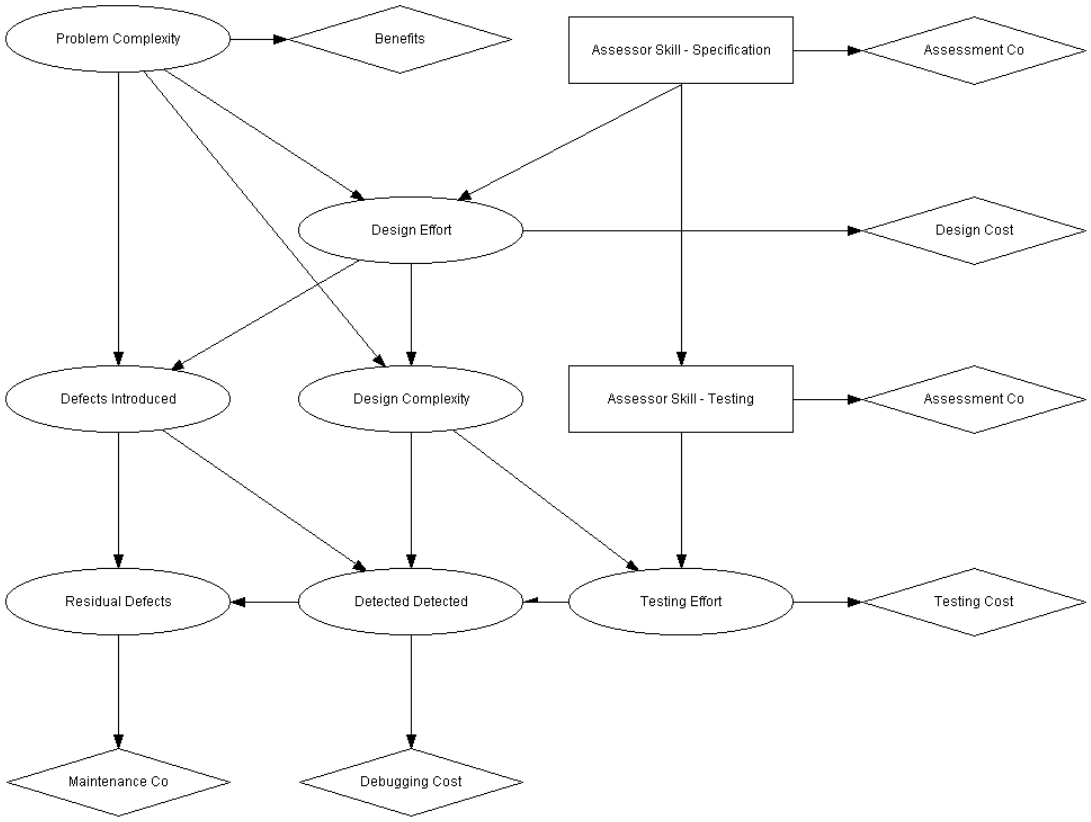

Figure A - BBN Topology

The topology of the Defect Density BBN is shown in Figure A. The ellipses represent 'chance' variables, the rectangles show the 'decisions', the diamonds represent 'utility' (cost/benefit) variables and the arrows show the flow of information or cause-effect links. The variables represented are measured on ordinal, subjective, scales. Subjective scales are used to make the model simpler; there is no theoretical impediment to modelling ratio scales and continuous variables. Each variable has the following states: very-high, high, medium, low, very low or none (optional for some variables). The probabilities attached to each of these states is determined from an analysis of the literature or common-sense assumptions about the direction and strength of relations between variables.

The BBN can be explained in two stages. The first stage covers the life-cycle processes of specification, design or coding and the second stage covers testing. In Figure A p*roblem complexity* represents the degree of complexity inherent in the set

of problems to be solved by development. We can think of these problems as being discrete functional requirements in the specification. Solving these problems accrues *benefits to* the user. At the specification stage a project manager assesses the complexity of the problems and assigns *design effort* accordingly. The skill with which this is done is denoted by the variable: *assessor skill—specification*. This assessment process could involve formal measurement, using function points for example, subjective judgement or some combination of both. Assessing the complexity of the problem accrues an *assessment cost—specification*. Any mis-match between the problem complexity and design effort is likely to cause the introduction of defects and a greater design complexity. Hence the arrows between *design effort, problem complexity*, *introduced defects* and *design complexity*. For example an optimistic project manager may allocate a small amount of design effort to a complex problem simply because the complexity was underestimated during assessment of the specification. Applying design effort incurs a *design cost*.

In Figure A the testing stage follows the design stage. Here *design complexity* is assessed by the project manager in order to gauge the amount of *testing effort* to allocate. This decision is represented by the *assessor skill—testing variable*. This is similar to the specification assessment process in that the project manager may measure the design complexity directly using appropriate static or dynamic metrics or will make a guess based on intuition and experience. The extent to which either of these measure precisely the actual design complexity will be uncertain. Doing the assessment will incur *assessment cost—testing*. Ideally any testing effort allocated would match that required by the design complexity. However in practice the testing effort actually allocated may be much less, whether by intent or accident. The mis-match between testing effort and design complexity will influence the number of *defects detected*, which is bounded by the number introduced. Fixing these defects during testing incurs a *de-bugging cost*. The difference between the defects detected and defects introduced is the *residual defects* count. Any residual defects will be released with the product and may increase the *maintenance costs*, incurred by the user and maintainer.

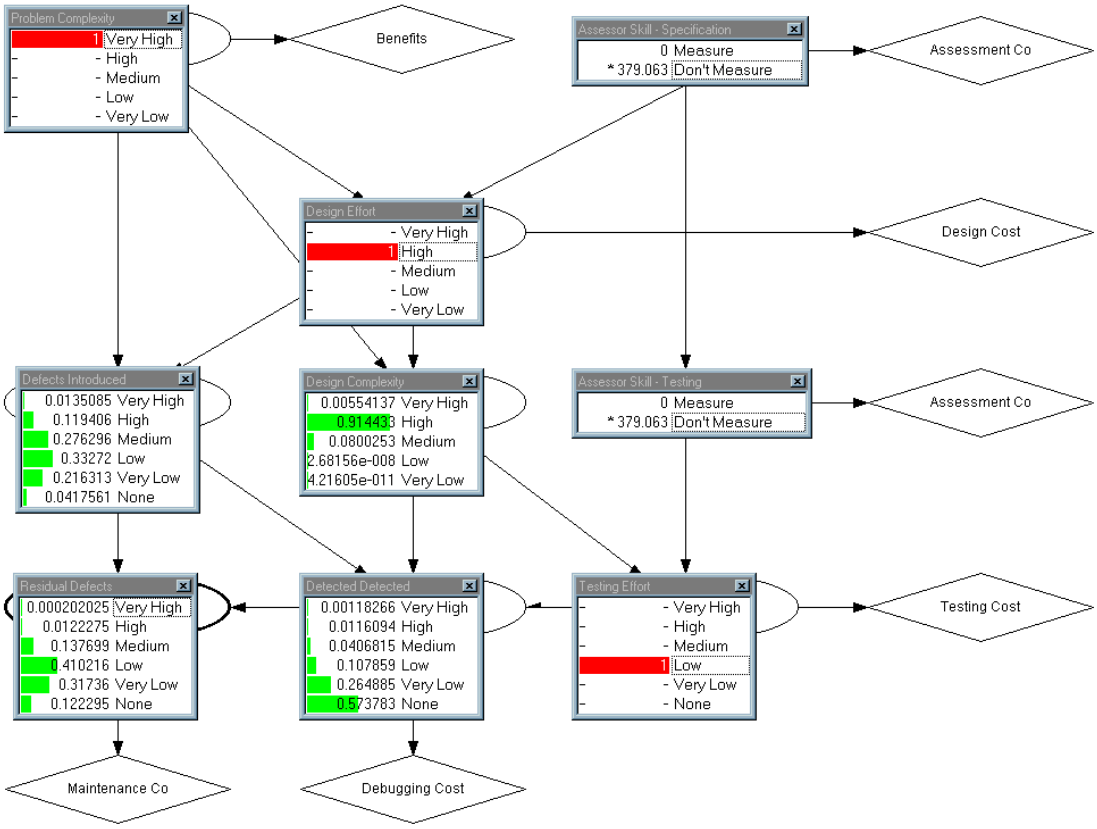

Figure B - Is Bigger Better? Dilemma

Figure B shows the execution of the defect density BBN model for the 'Is Bigger Better?' dilemma using the *Hugin Explorer* tool. Each of the decision and chance variables is shown as a window with a histogram of the predictions made based on the facts entered. The scenario runs as follows. A very complex problem is represented as a fact set at 'very high'. Assume the project manager performs no precise estimation on this so the *assessment skill—specification* variable is set to 'no measurement'. This results in the allocation of 'high' *design effort*, rather than 'very high' commensurate with the problem complexity. The model then propagates these 'facts' and predicts the *design complexity* with a peak at 'high' with probability of approx. 90%. The *introduced defects* follows a modal distribution shape with a peak at 'medium' with probability of around 27%. We may also find that the project manger is again optimistic. He does not measure the *design complexity* and allocates a 'low' level of *testing effort*. This results in low levels of *defects detected*, with approximately 60% probability of finding no defects at all. From the predicted values for detected and introduced defects is propagated to predict the residual defects. Residual defects peaks at 'low' with around 40% probability but with a significant tail towards medium and high numbers of residual defects.

From the model we can see a credible explanation for observing large 'modules' with lower defect densities. Under allocation of design effort for complex problems results in more introduced defects and higher design complexity. Higher design complexity requires more testing effort, which is unavailable, leading to less defects being discovered than are actually there. Dividing the small detected defect counts with large design complexity values will result in small defect densities! The model

explains the “is bigger better” phenomena without ad-hoc explanation or identification of ‘outliers’.

5. The Way Forward

At a general level we can see how the use of BBNs and the defect density model provide a significant new approach to modelling software engineering processes and artefacts. The dynamic nature of this model provides a way of simulating different events and identifying optimum courses of action based on uncertain knowledge. These benefits are reinforced when we examine how the model explains known results, in particular the ‘Is Bigger Better?’ dilemma. Our new approach shows how we can build complex webs of interconnection between process, product and resource factors in a way hitherto unachievable. We also should how we can integrate uncertainty and subjective criteria into the model without sacrificing rigour and illustrate how decision-making throughout the development process influences the quality achieved.

The benefits of this new approach are:

- it is more useful for project management than outlier analysis and classical statistics
- it incorporates current research ideas and experience
- it can be used to train managers and enable comparison of different decisions by simulation and what-if analyses
- it integrates a form of cost and quality forecasting

So far we have explained historical results rather than real projects. Much work remains to be done to:

- provide guidelines on how to apply the approach to specific situations
- develop a modular approach where whole development processes can be modelled using linked BBNs
- assess the validity of the model by testing its predictions on real projects

We have embarked on the above tasks in the area of safety cases in the CEC ESPRIT project SERENE (Safety and Risk Evaluation using Bayesian Nets) and will be improving it for statistical software process control in the IMPRESS (Improving the Software Process using Bayesian Nets) project funded by UK EPSRC. We will be applying the defect density BBN model to a project with Ericsson Radio Systems in Sweden and are working with the UK Defence Research Agency (DRA) to develop BBNs for procurement processes.

Acknowledgements

This work was supported in part by the ESPRIT projects SERENE and DEVA.

References

[Akiyama 1971] Akiyama, F 'An example of software system debugging', Inf Processing 71, 353-379 1971

[Bache and Bazzana 1993] R.Bache, G.Bazzana (1993) Software metrics for product assessment , McGraw Hill, London.

[Basili and Perricone 1984] V.R. Basili and B.T. Perricone, "Software Errors and Complexity: An Empirical Investigation", Communications of the ACM, 1984, pp.42-52.

[Compton and Withrow 1990] , 'Prediction and control of Ada software defects', Proc 2nd Annual Oregon Workshop on Software Metrics, March 1990.

[Cusumano 1991] Cusumano, MA, Japan's Software Factories, Oxford University Press, 1991.

[Fenton and Pfleeger 1996] Fenton NE, Pfleeger SL Software Metrics: A Rigorous and Practical Approach. International Thomson Computer Press, 1996.

[Gaffney 1984] J.E.Gaffney, JR., "Estimating the Number of Defects in Code", IEEE Trans. Software Engineering, Vol.SE-10, No.4, 1984

[Hatton 1994]Hatton, L. (1994). C and Safety Related Software Development: Standards, Subsets, testing, Metrics, Legal issues. McGraw-Hill.

[Hugin 1996] Hugin Expert A/S Niels Jernes Vej 10 DK - 9220 Aalborg, Denmark

[Koga 1992] Koga K, 'Software Reliability Design Method in Hitachi', Proceedings of the 3rd European Conference on Software Quality, Madrid, 1992

[Lipow 1982] M.Lipow, "Number of Defects per Line of CODE", IEEE Trans. Software Engineering, Vol.SE-8, No.4, 1982, 437-439, 1982

[Moeller and Paulish 1993] K.H. Moeller, D. Paulish, 'An emprirical investigation of software defect distribution', Proc 1st Intl Software Metrics Symp, IEEE CS Press, 82-90, 1993

[Neil 1992] Neil, M.D. "Multivariate Assessment of Software Products". Journal of Software Testing, Verification and Reliability, Vol 1(4), pp 17-37, 1992.

[Ottenstein 1979] Ottenstein LM, 'Quantitative estimates of debugging requirements', IEEE TSE 5(5), 504-514, 1979

[Pearl 1988] Pearl J, Probabilistic reasoning in intelligent systems, Morgan Kaufmann, Palo Alto, CA, 1988.

[Shen et al 1983] Shen VY, Conte SD, Dunsmore H, 'Software science revisited: a critical analysis of the theory and its empirical support', IEEE Trans Soft Eng, SE-9(2), 1983, 155-165.

[Shen 1985] Shen VY, Yu T, Thebaut SM, Paulsen LR, 'Identifying error-prone software - an empirical study', IEEE Trans Soft Eng SE-11(4) 1985, 317-323.

[Veevers and Marshall 1994] Veevers A and Marshall AC, A relationship betwen software coverage metrics and reliability', J Software Testing, Verification and Reliability, 4, 3-8, 1994

[Yasuda 1989] Yasuda, K.'Software Quality Assurance Activities in Japan. In Japanese Perspectives in Software Engineering,.187-205, Addison-Wesley, 1989.

[Friedman and Voas, 1995] M. A. Friedman and J. M. Voas Software Assessment: Reliability, Safety and Testability. John Wiley and Sons, 1995.