Software Metrics and Risk

FESMA 99
2nd European Software Measurement Conference
8 October, 1999

Norman Fenton and Martin Neil
Agena Ltd
and
Centre for Software Reliability
City University,
email: norman@agena.co.uk

Abstract

Most software metrics activities are carried out for the purposes of risk analysis of some form or another. Yet traditional metrics approaches, such as regression-based models for cost estimation and defects prediction, provide little support for managers wishing to use measurement to analyse and minimise risk. Many traditional approaches are not only insufficient in this respect but also fundamentally flawed. Significant improvements can be achieved by using causal models that require no new metrics. The new approach, using Bayesian nets, provides true decision-support and risk analysis potential.

1 Introduction

Among the many claimed benefits of software metrics, the most significant is that they are supposed to provide information to support managerial decision-making during the software lifecycle. Good support for decision-making implies support for risk analysis. Unfortunately, most metrics approaches do not provide such support. In this paper we focus on the use of software metrics for:

1. defect modelling/prediction
2. resource modelling/prediction

These two applications have, to a very large extent, driven the entire software metrics subject area. The key in both cases has been the assumption that good 'size' measures should drive any predictive models. In Section 2 we summarise the basic approaches in each case and highlight their weaknesses, with recent empirical examples. In Section 3 we discuss the need for a causal approach to software prediction and introduce the notion of Bayesian nets. In Sections 4 and 5 we provide examples of causal models (using Bayesian nets) for defect prediction and resource prediction respectively.

2 The classic approaches to defect and resource prediction

Most early work on software metrics focused on the Lines of Code measure (LOC or KLOC for thousands of lines of code). This was used routinely as the basis for measuring both programmer productivity (LOC per programmer month) and program quality (defects per KLOC). In other words LOC was being used as a surrogate measure of different notions of program size. The early resource prediction models (such as those of [Putnam 1978] and [Boehm 1981]) also used LOC or related metrics like delivered source instructions as the key size variable. In 1971 Akiyama [Akiyama 1971] published what we believe was the first attempt to use metrics for software quality prediction when he proposed a crude regression-based model for module defect density (number of defects per KLOC) in terms of the module size measured in KLOC. In other words he was using KLOC as a surrogate

measure for program complexity.

The obvious drawbacks of using such a crude measure as LOC as a surrogate measure for such different notions of program size such as effort, functionality, and complexity, were recognised in the mid-1970's. The need for more discriminating measures became especially urgent with the increasing diversity of programming languages. After all, a LOC in an assembly language is not comparable in effort, functionality, or complexity to a LOC in a high-level language. Thus, the decade starting from the mid-1970's saw an explosion of interest in measures of software complexity (pioneered by the likes of [Halstead 1977] and [McCabe 1976]) and measures of size (such as function points pioneered by [Albrecht 1979] and later by [Symons 1991]) which were intended to be independent of programming language.

Despite these advances the approaches to both defects prediction and resource prediction have remained fundamentally unchanged since the early 1980's. Figure 1 provides a schematic view of the classical approach to defect prediction (as seen, for example in [Basili and Preicon 1984], [Compton and Withrow 1990], [Gaffney 1984], [Lipow 1982], [Shen et al 1985]. Size (whether it be solution size, as in complexity metrics or LOC-based approaches, or problem size, as in function point based approaches) is the key driver. In many cases it is the only driver, although recently resource and process quality factors have been considered.

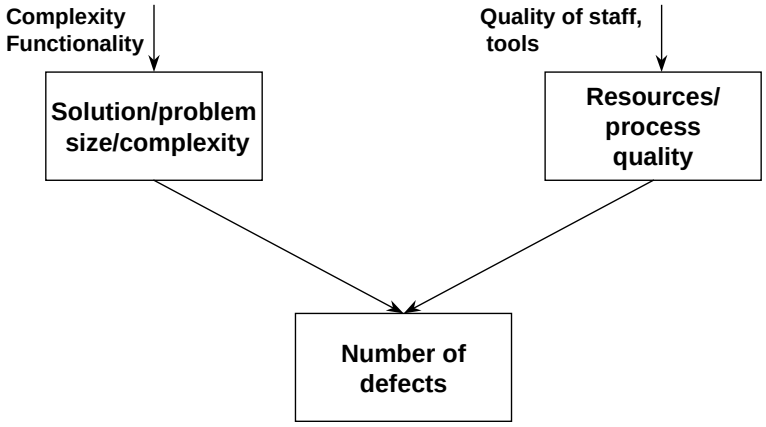

Figure 1 Classical approach to defect modelling

In [Fenton and Neil 1999] we provided an extensive critique of this classical approach. We identified problems such as:

- Fails to distinguish different notions of 'defect'
- Statistical methods are often flawed
- Size is wrongly assumed to be a causal factors for defects
- Obvious causal factors are not modelled
- Black box models hide crucial assumptions
- Cannot handle uncertainty
- Resulting models provide little support for risk assessment and reduction

Figures 2-5 show data from a recent major case study [Fenton and Ohlson 1999] that highlighted some of these fundamental problems. In each figure the dots represent modules sampled at random from a very large system. Figure 2 (in which 'faults' refers to all known faults discovered pre-and post-delivery) confirms what many studies have shown: module size is correlated with number of faults, but is not a good predictor of it. Figure 3 shows that complexity metrics, such as cyclomatic complexity, are not significantly better (and in any case are very strongly correlated to LOC). This is true even when we separate out pre- and post-delivery faults.

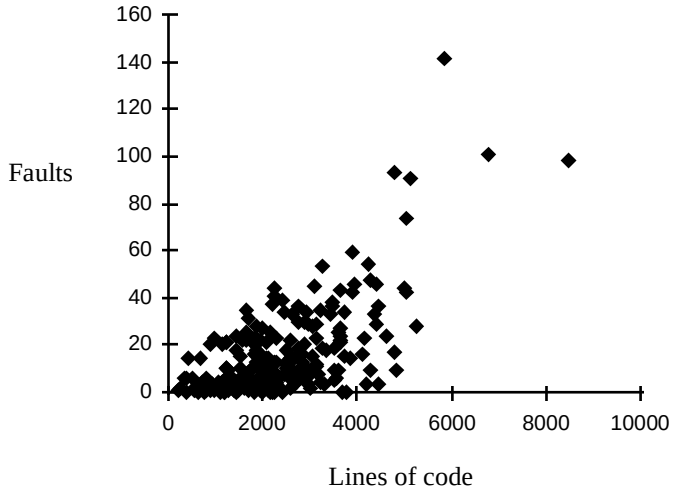

Figure 2 Scatterplot of LOC against all faults for major system (each dot represents a module).

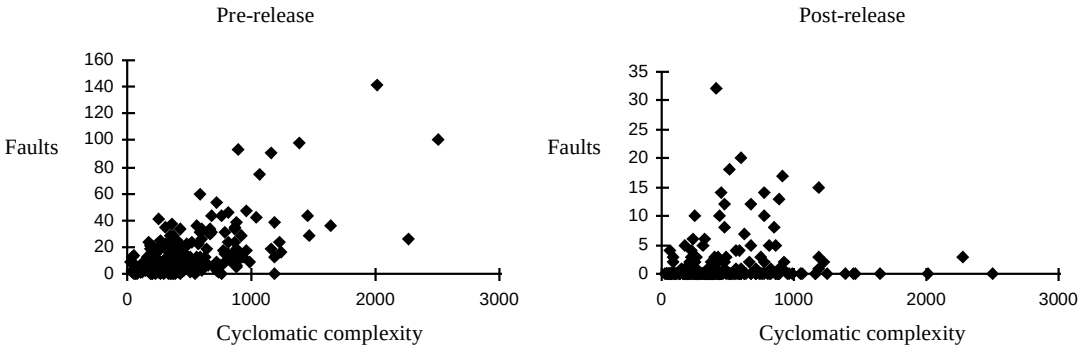

Figure 3 Scatterplots of cyclomatic complexity against number of pre-and post-release faults for release *n+1* (each dot represents a module).

Figure 4 exposes the myth that simple size metrics might be good predictors of the normalised measure of faults - *fault density* (faults per LOC). This myth is actually enshrined in much of software engineering, with the notion that smaller modules are less 'fault-prone' than large ones. The evidence shows a random relationship.

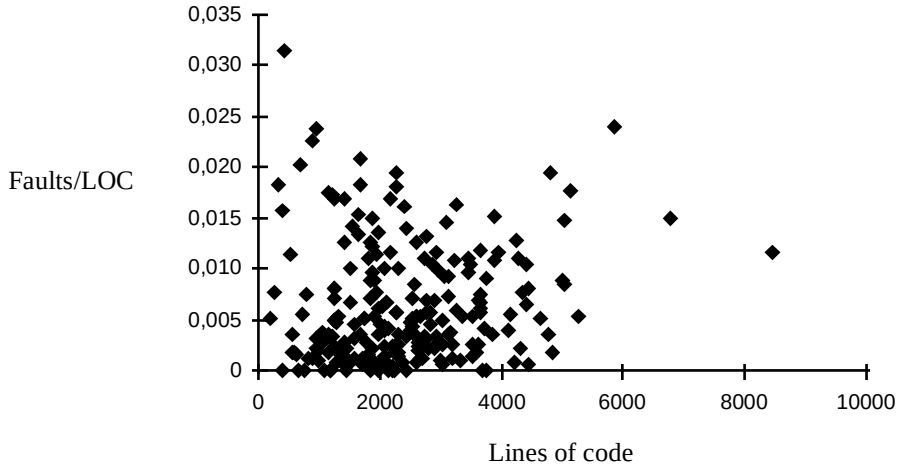

Figure 4 Scatter plot of module fault density against size (each dot represents a module).

Most devastating of all for the classical approach to defect modelling is the result shown in Figure 5. This result (which was repeated in other systems) demolishes the popular hypothesis that the number of pre-release faults is a good indicator of the number of post-release faults (the latter being the number you really are most interested in predicting). In fact, there is strong evidence that modules which are very fault-prone pre-release are likely to reveal very few faults post-release. Conversely, the truly 'fault-prone' modules post-release are the ones which revealed no faults pre-release. Why are these results devastating for much classical fault prediction models? Because many of those models were 'validated' on the basis of using pre-release fault counts as a surrogate measure for operational quality.

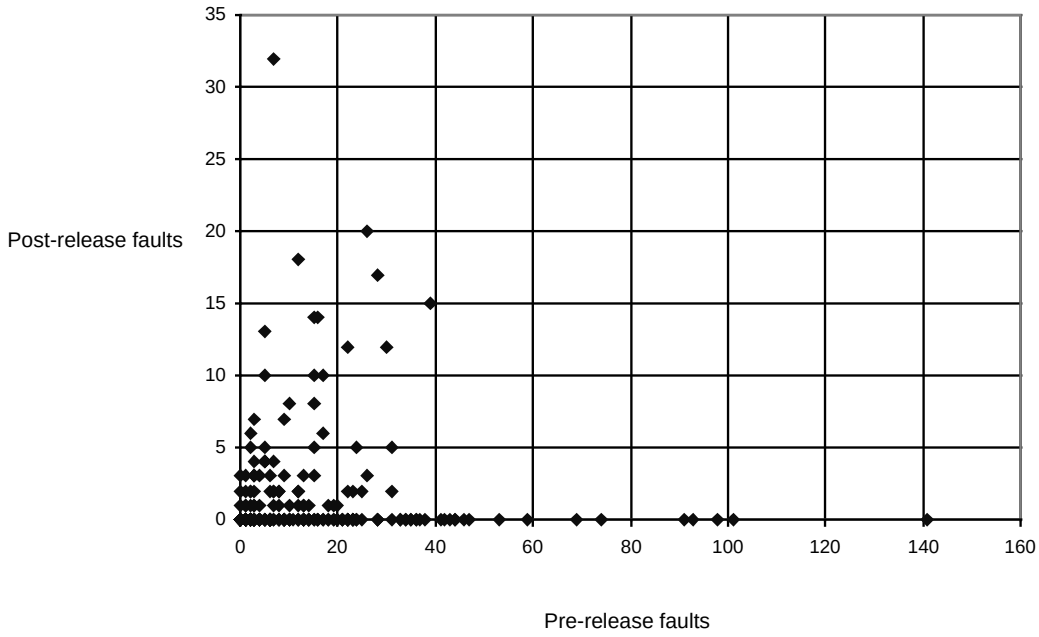

Figure 5 Scatter plot of pre-release faults against post-release faults for a major system (each dot represents a module)

There are, of course, very simple explanations for the phenomenon observed in Figure 5. Most of the modules that had high number of pre-release, low number of post-release faults just happened to be very well tested. The amount of testing is therefore a very simple explanatory factor that must be incorporated into any predictive model of defects. Similarly, a module that is simply never executed in operation, will reveal no faults no matter how many are latent. Hence, operational usage is another obvious explanatory factor that must be incorporated.

The absence of any causal factors that explain variation is a feature also of the classic approach to resource prediction that is shown in Figure 6.

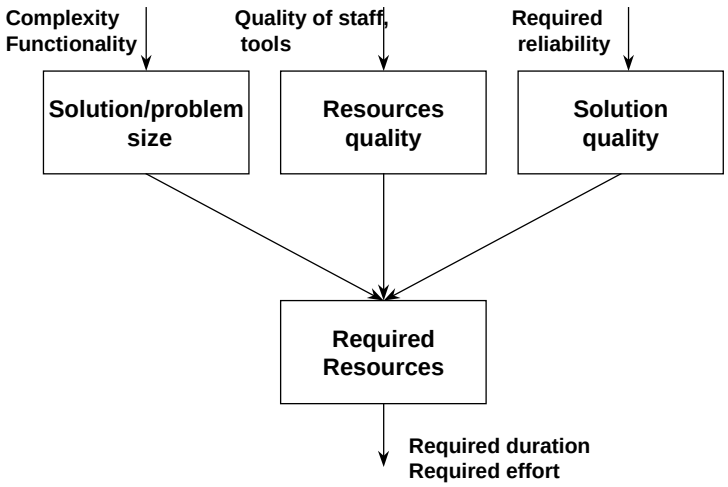

Figure 6 Classic approach to resource prediction

The basic problems with this approach are:

- They are regression-based models based on limited historical data of projects that just happened to have been completed. Based on typical software projects these are likely to have produced products of poor quality. T is therefore difficult to interpret what a figure for effort prediction based on such a model actually means.
- It fails to incorporate any true causal relationships, relying often on the fundamentally flawed assumption that somehow the solution size can influence the amount of resources required. This defeats the economic definition of a production model where output = f(input) rather input = f(output).
- There is a flawed assumption that projects do not have prior resourcing constraints. In practice all projects do, but these cannot be accommodated in the models. Hence the 'prediction' is premised on impossible assumptions and provides little more than a negotiating tool.
- Black box models hide crucial assumptions
- Cannot handle uncertainty

To provide genuine risk assessment and decision support for managers we need to provide the following kinds of predictions:

- For a problem of this size, and given these limited resources, how likely am I to achieve a product of suitable quality?
- How much can I scale down the resources if I am prepared to put up with a product of specified lesser quality?
- The model predicts that I need 4 people over 2 years to build a system of this kind of size. But I only have funding for 3 people over one year. If I cannot sacrifice quality, how good do the staff have to be to build the systems with the limited resources?

Our aim is to show that causal models, using Bayesian nets can provide relevant predictions, as well as incorporating the inevitable uncertainty, reliance on expert judgement, and incomplete information that are pervasive in software engineering.

3 A causal approach: Bayesian Belief Nets (BBNs)

The challenge is to produce models of the software development and testing process which take account of the crucial concepts missing from classical statistical back-box approaches. Specifically we need models that can handle:

- diverse process and product variables;
- empirical evidence *and* expert judgement;
- genuine cause and effect relationships;
- uncertainty;
- incomplete information.

At the same time the models must not introduce any additional metrics overheads, either in terms of the amount of data-collection or the sophistication of the metrics. After extensive investigations during the DATUM project 1993-1996 into the range of suitable formalisms [Fenton et al 1998] we concluded that Bayesian belief nets (BBNs) were by far the best solution for our problem. The only remotely relevant approach we found in the software engineering literature was the process simulation method of [Abdel-Hamid 1991], but this did not attempt to model the crucial notion of uncertainty.

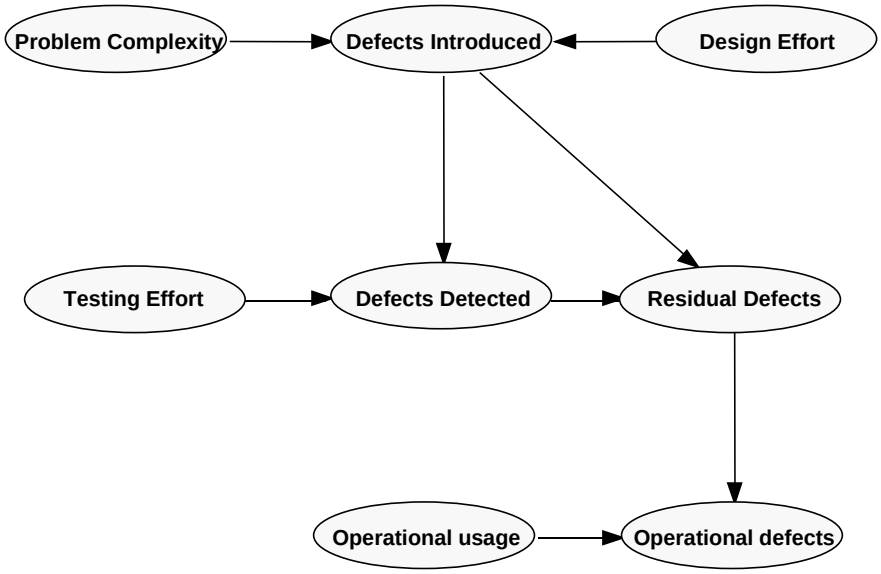

Figure 7 Defects BBN (simplified)

A BBN is a graphical network (such as that shown in Figure 7) together with an associated set of probability tables. The nodes represent uncertain variables and the arcs represent the causal/relevance relationships between the variables. The probability tables for each node provide the probabilities of each state of the variable for that node. For nodes without parents these are just the marginal probabilities while for nodes with parents these are conditional probabilities for each combination of parent state values.

Although the underlying theory (Bayesian probability) has been around for a long time, building and executing realistic BBN models has only been made possible because of recent algorithms (see [Jensen 1996]) and software tools that implement them [Hugin A/S]. To date BBNs have proven useful in practical applications such as medical diagnosis and diagnosis of mechanical failures. Their most celebrated recent use has been by Microsoft where BBNs underlie the help wizards in Microsoft Office.

CSR, in a number of research and development projects since 1992, has pioneered the use of BBNs in the broad area of assessing dependability of systems. For example, in recent collaborative projects we have used BBNs to:

- provide safety or reliability arguments for critical computer systems (the DATUM, SHIP, DeVa and SERENE projects have all addressed this problem from different industrial perspectives [Courtois et al 1998, Delic at al 1997, Fenton et al 1998, SERENE 1999]);
- provide improved reliability predictions of prototype military vehicles (the TRACS project, [TRACS 1999]);
- predict general software quality attributes such as defect-density and cost (the IMPRESS project [Fenton and Neil 1999, Lewis et al 1998]).

In consultancy projects we have used BBNs to

- assess safety of PES components in the railway industry;
- provide predictions of insurance risk and operational risk;
- predict defect counts for software modules in consumer electronics products.

The BBN in Figure 7 is a simplified version of the last example. Like all BBNs the probability tables were built using a mixture of empirical data and expert judgements. In the next section we describe this BBN and show how the model is executed and applied using the Hugin tool.

4 Defects model BBN

Like any BBN, the model in Figure 7 contains a mixture of variables we might have evidence about and variables we are interested in predicting. At different times during development and testing different information will be available, although some variables such as 'number of defects introduced' will never be known with certainty. With BBNs, it is possible to propagate consistently the impact of evidence on the probabilities of uncertain outcomes. For example, suppose we have evidence about the number of defects discovered in testing. When we enter this evidence all the probabilities in the entire net are updated

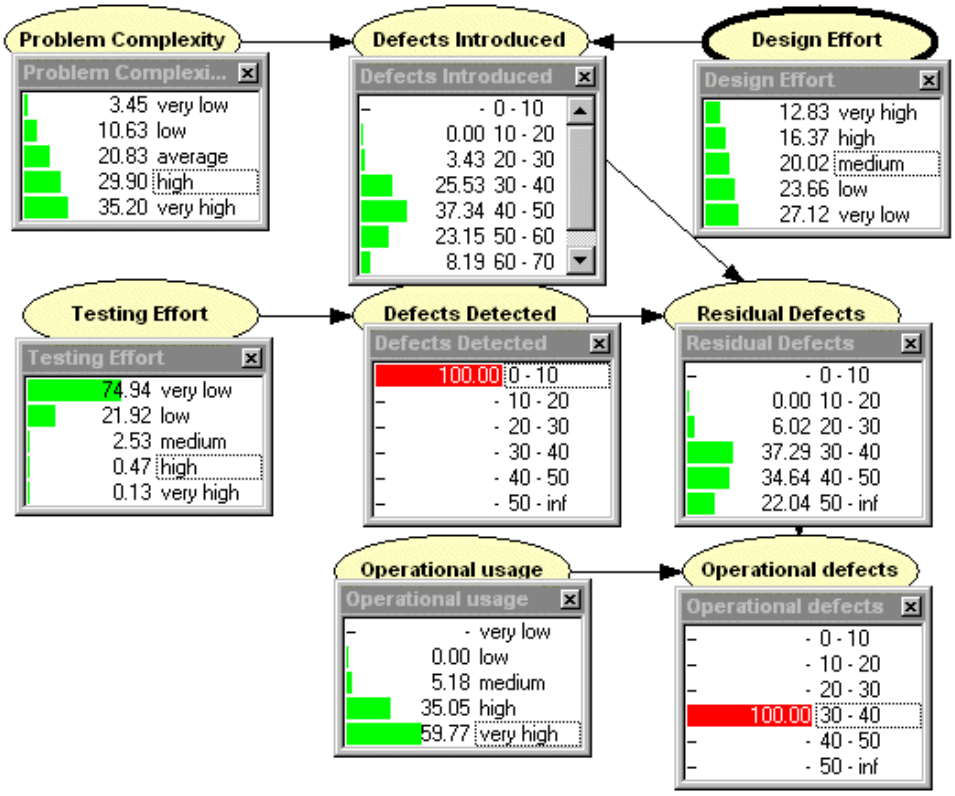

Figure 8 State of BBN probabilities showing a scenario of many post-release faults and few pre-release faults

Thus, **Figure 8** explores the common empirical scenario (highlighted in Section 2) of a module with few pre-release defects (less than 10) and many post-release (between 30 and 40). Having entered this evidence, the remaining probability distributions are updated. The result *explains* this scenario by showing that it was very likely that a 'very low' amount of testing was done, while the operational usage is likely to be 'very high'. The problem complexity is also more likely to be high than low.

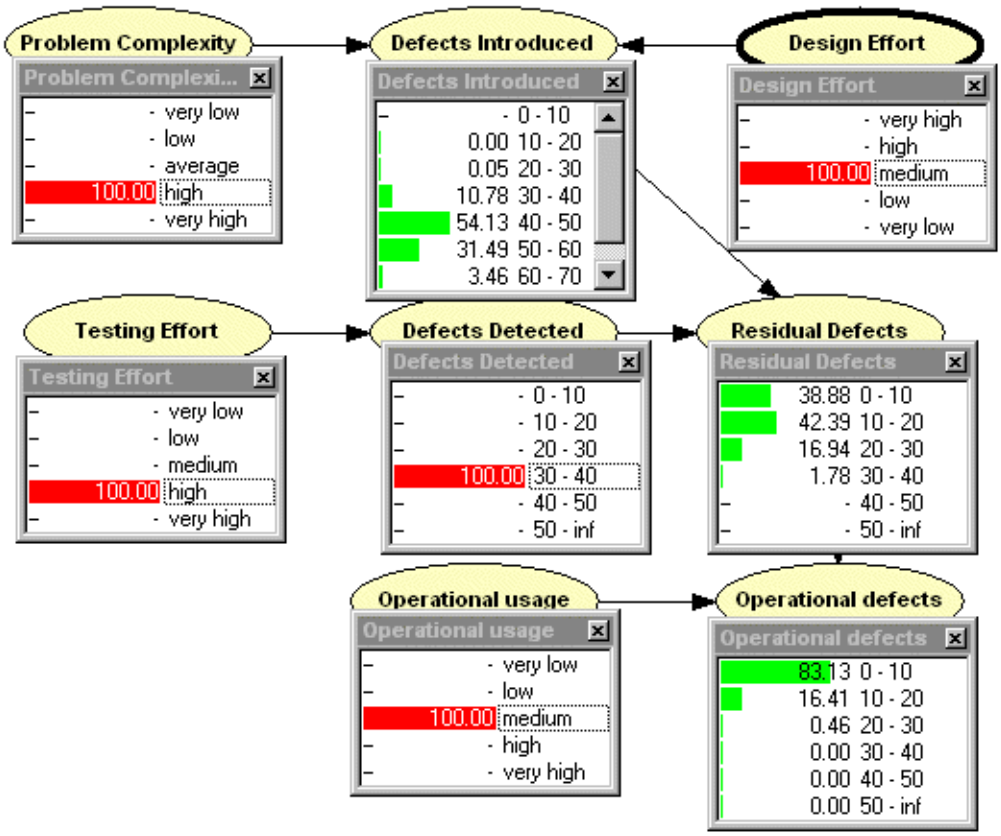

Figure 9 State of BBN probabilities showing a scenario of many pre-release faults and few post-release faults

Figure 9 shows the effects of entering different types of evidence. In this case the model will actually *predict* the empirical scenario of a module with many pre-release defects but few post-release defects. In this case we have a module of 'high' complexity for which we used 'medium' design effort and 'high' testing effort. The actual number of defects found in testing is in the interval 30-40. The prediction for number of operational defects, given 'medium' operational usage is a distribution with a mean in the region of 5. This compares with a prediction for number of residual defects having a mean around 13. No classical model can make this kind of prediction

The beauty of BBNs is that we can enter as many or as few observations anywhere in the net and still get predictions for all of the variables. It is in this respect that BBNs provide true decision support for risk assessment. For example, suppose we know only that the module is complex and that it will be used extensively. We have a *requirement* that the module will have less than 10 defects. We would like to know in this case how much testing effort and design effort is required. Figure 10 shows the results. For example, in this scenario the probability that *testing effort* can be less than 'medium' is extremely low (less than 0.05).

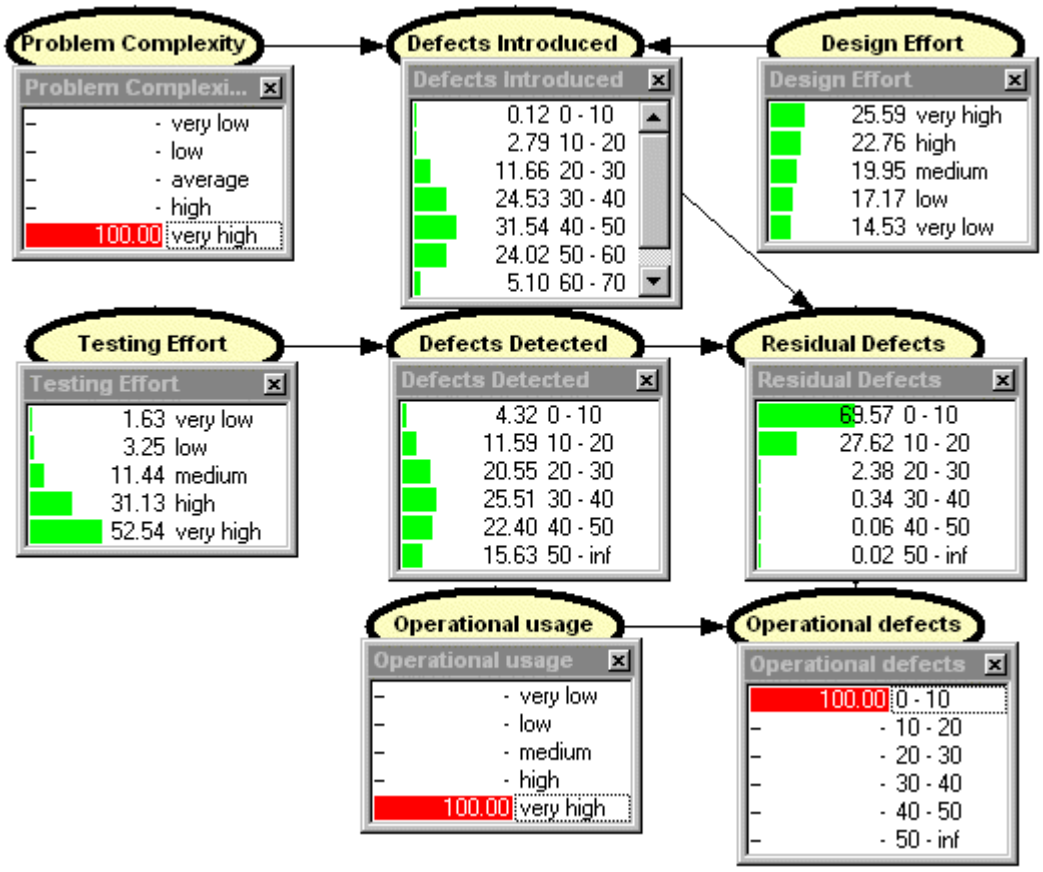

Figure 10: Low number of post-release defects as a requirement

5 Resource model BBN

A sensible BBN model for software resource estimation inevitably involves a fairly large number of variables. However, conceptually the model is quite simple and is shown in Figure 11.

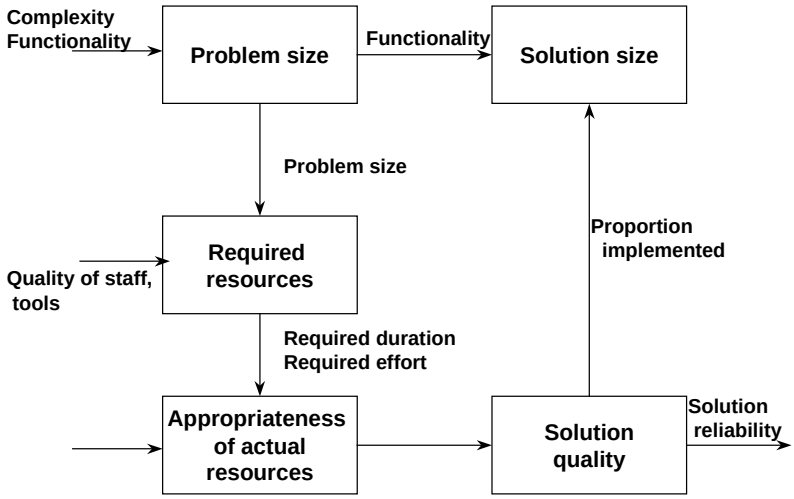

Figure 11 Causal model for software resources

Each of the boxes in Figure 11 represent subnets. For example, there is a subnet concerned with *problem size*, which contains variables such as *functionality* and *complexity*. *Problem size* influences both *solution size* and *required resources*. The subnets that make this kind of model so different from

the classical approach are the subnets *appropriateness of actual resources* and *solution quality.* The basic idea is that the *required resources* for a given problem are always some kind of an ideal based on best practice. What matters are how well these ideal resources match the actual resources available. This match (which is calculated in the *appropriateness of actual resources* subnet) determines the solution quality.

The details of each subnet are as follows:

5.1 "Problem Size" subnet

This is shown in Figure 12. In this case problem size is defined in terms of *functionality* and *complexity.* In practice any measure of functionality could be used, but in the example we have chosen to use raw function points. Problem size is like an adjusted function count measure where high complexity increases the count and low complexity decreases it.

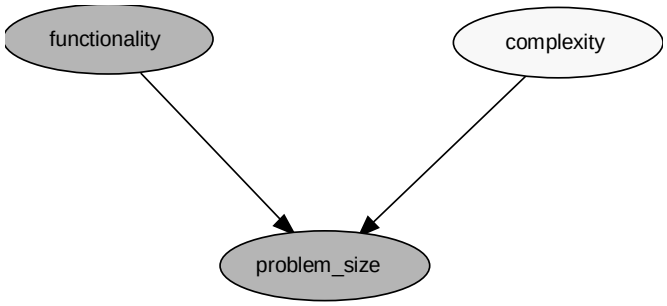

Figure 12 Subnet: Problem size

5.2 "Required Resources" subnet

This is shown in Figure 13. In this net the potential_productivity_rate distribution given problem size is based on empirical data. The adjusted_productivity rate takes account of resource quality factors. The required effort and required duration distributions are based on a range of published empirical studies.

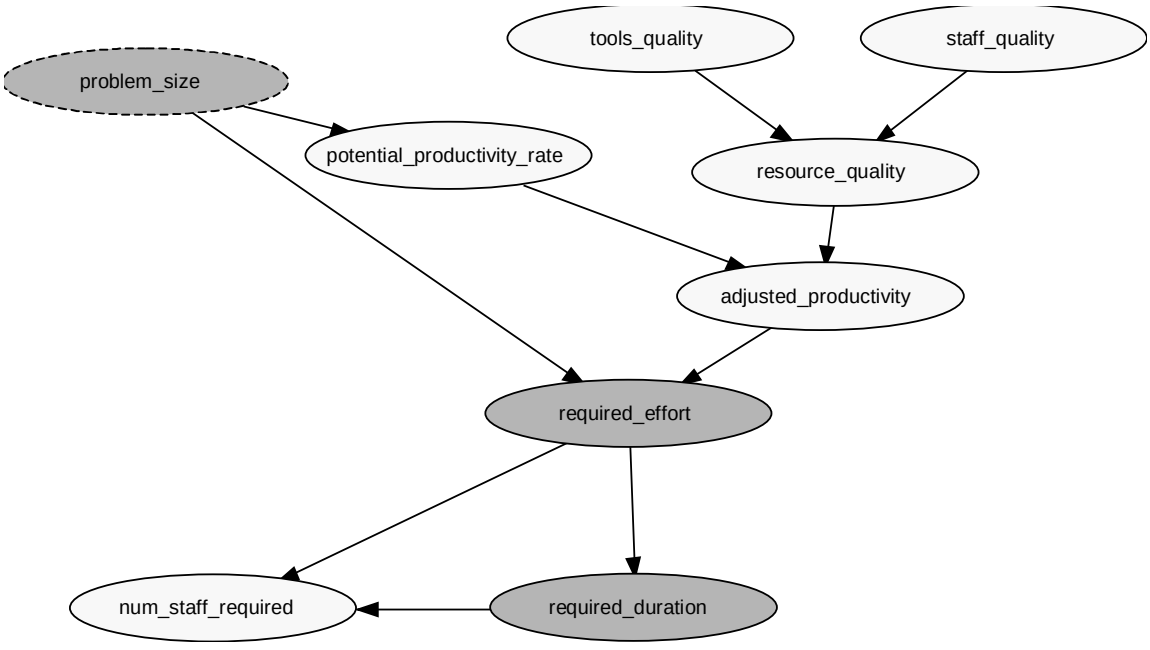

Figure 13 Subnet "Required resources"

5.3 "Appropriateness of resources" subnet

This is shown in Figure 14. In this we calculate the appropriateness of effort (resp duration) by comparing the required effort (resp duration) with the actual effort.

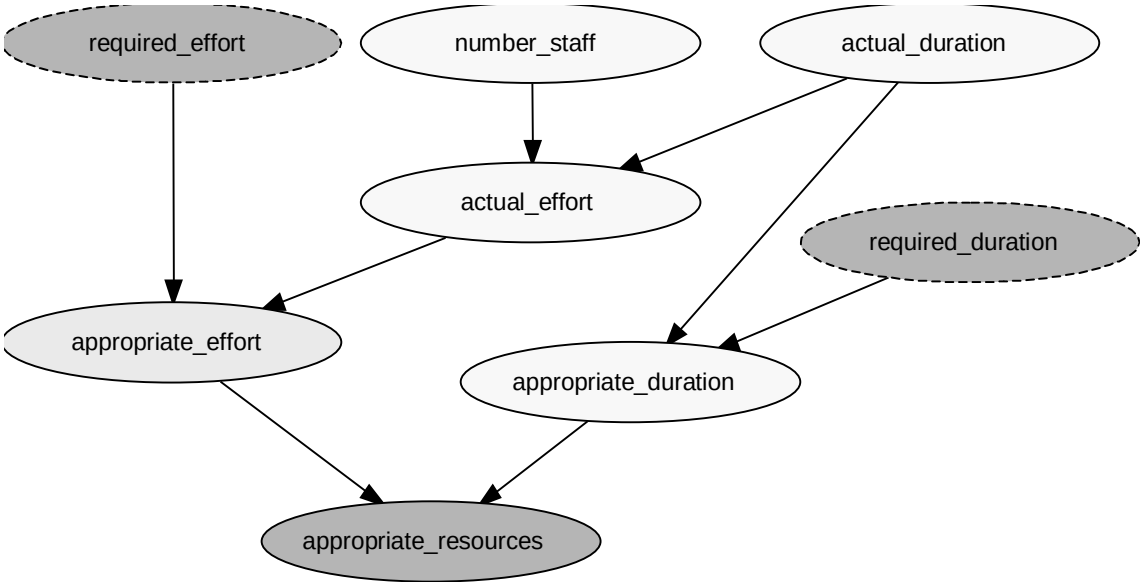

Figure 14 Subnet "Appropriateness of resources"

5.4 "Accuracy of implementation" subnet

This is shown in Figure 15. The node *accuracy_implemented* can be though of as a measure of the operational reliability of the product (we use a scale 0 to 100 where the figure represents 'percentage accuracy'). The node *proportion_func_implemented* represents the proportion of specified functionality that is actually implemented. This subnet really introduces the 'real world' to resource modelling by recognising that anything other than maximum resources will lead to some trade-off: either the accuracy or the amount of functionality implemented will suffer (or both). Inappropriately low resources combined with a hard requirement for perfect accuracy inevitably means less than 100% functionality implemented.

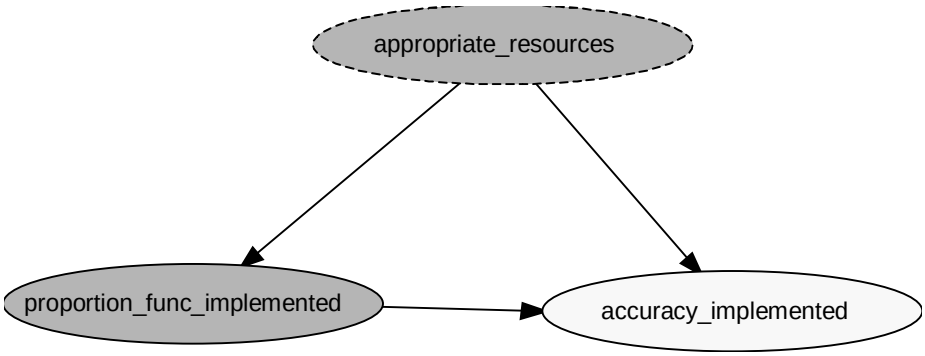

Figure 15 Subnet "Accuracy of implementation"

5.5 "Lines of Code" Subnet

This is shown in Figure 16. This has really only been included to show the true role of a solution size measure like KLOC. Clearly the solution size is influenced by the specified functionality and the

proportion of this functionality that is actually implemented. As solution size is measured in function points we simply apply a transformation (given different languages) using published empirical relationships between function points and KLOC.

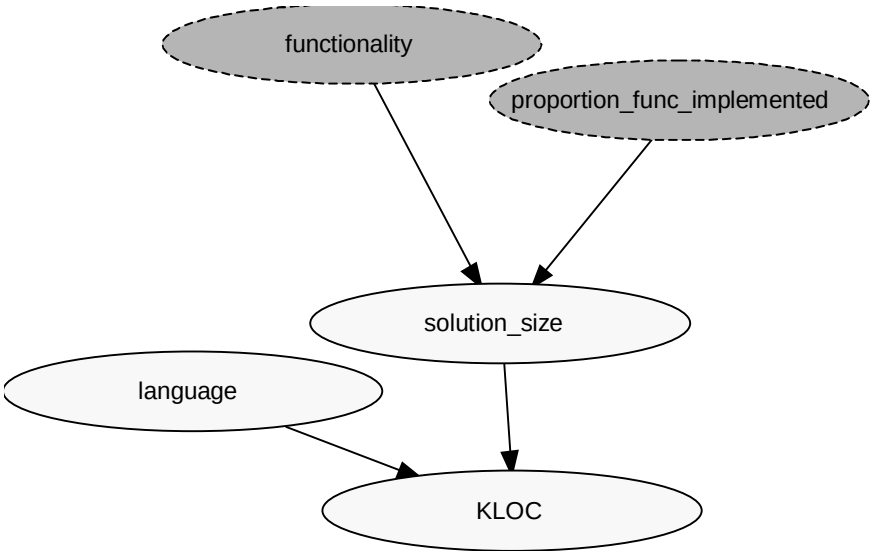

Figure 16 Subnet for "Lines of Code"

5.6 Using the BBN for risk assessment and prediction

We next give some illustrations of the BBN in use. Because of the sheer size of the complete BBN we are unable to show it one simple screen. Instead we select just a few monitors showing the probability distributions for some key variables.

Figure 17 shows what happens when we enter only the values for functionality (1400-1500 function points) and complexity (average). Without any other information the key variables have fairly wide distributions (as you should expect). For example, the required effort distribution has a mean around 80 person months but a wide variance, while the number of staff has a mean of around 4. Hence the required duration has a mean around 20 months. Most importantly note how flat the distribution is for ‘accuracy’. This confirms the empirical evidence that typical projects result in not especially reliable software. However, we believe that we are likely to implement at least 70% of the functionality (probability 0.75).

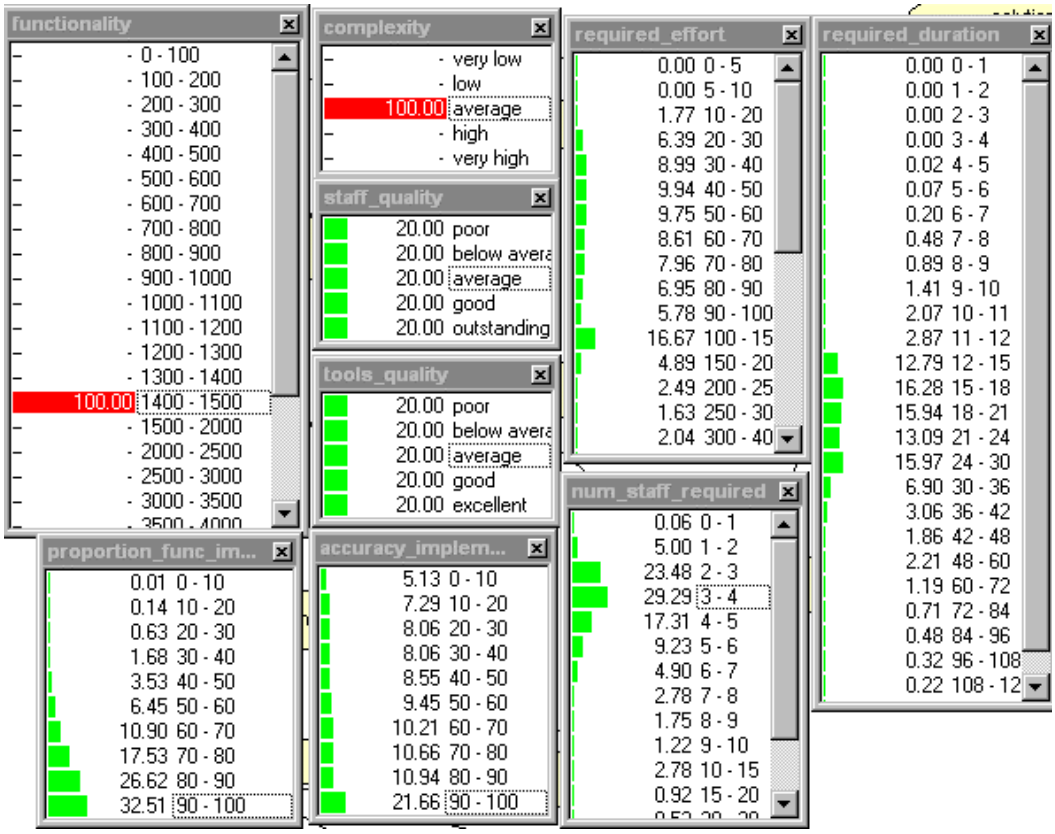

Figure 17 Resource BBN: Some of the probability distributions for specific values of problem size

Figure 18 shows what happened when we insert the requirements of perfect accuracy and complete implementation. The distribution for required effort move upwards, but note especially how the distribution of staff quality shifts significantly towards ‘outstanding’. In other words irrespective of the actual person months you put in you are only likely to achieve your requirements with very good staff.

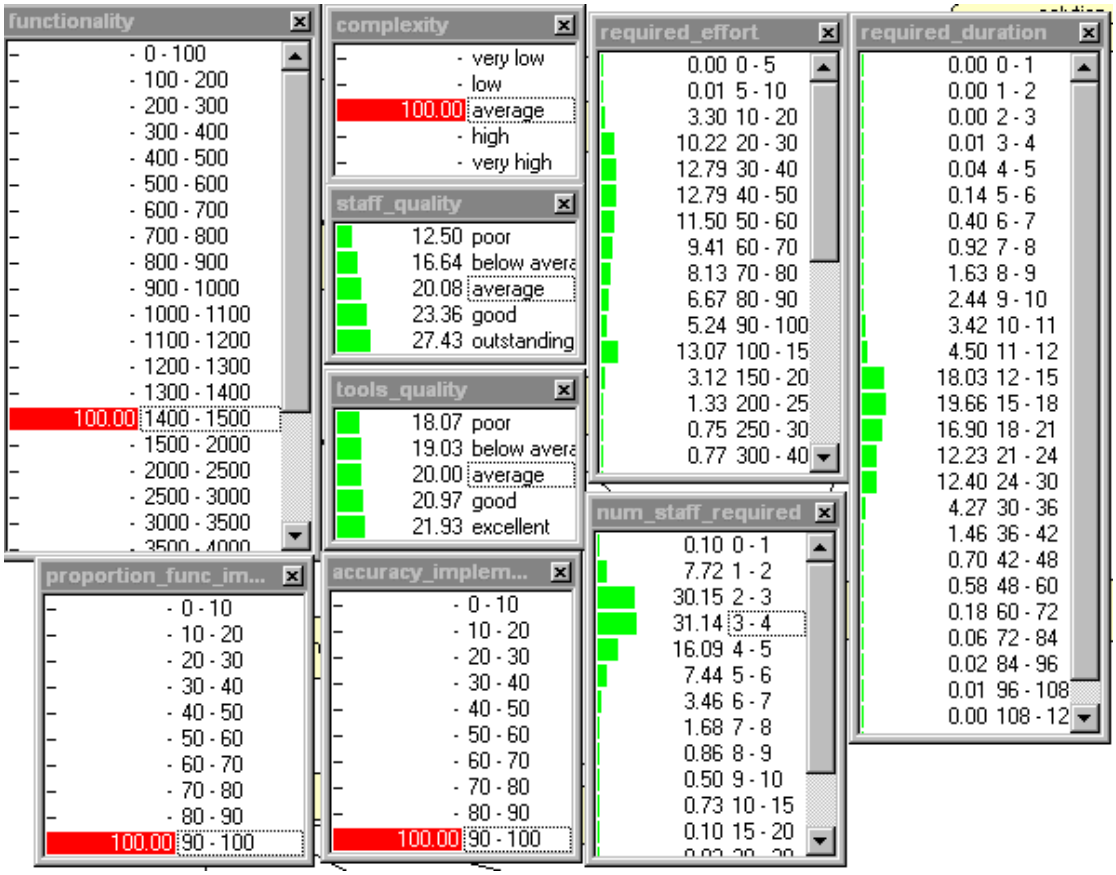

Figure 18 How the probabilities change when we 'require' high accuracy

This notion is really brought home in Figure 19. Here we have entered actual values for resources, namely that we have two staff working for 9-10 months. Because this is much less than the mean of the previously predicted required effort the effects are two-fold: on the one hand the required effort decreases because we are more likely to be believe that the original estimates were too high. However, most importantly, we see the distribution of *staff quality* skewed massively toward 'outstanding'. In other words, your only chance to achieve such demanding targets is a) to have truly outstanding staff; and b) to hope that the problem size was not as great as you first thought.

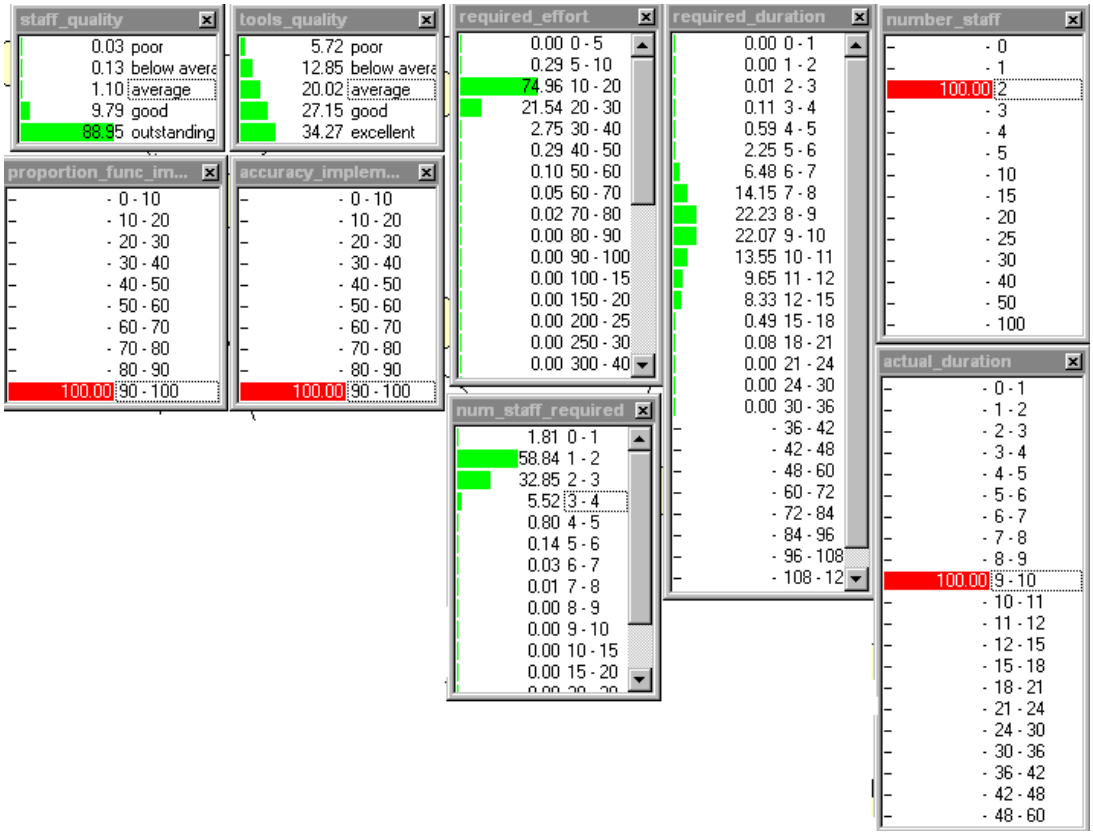

Figure 19 The new probabilities when we enter actual resource information

Figure 20 shows what happens when we enter actual observations about resource quality. If our staff and tools are 'average' then the accuracy of the implementation suffers dramatically.

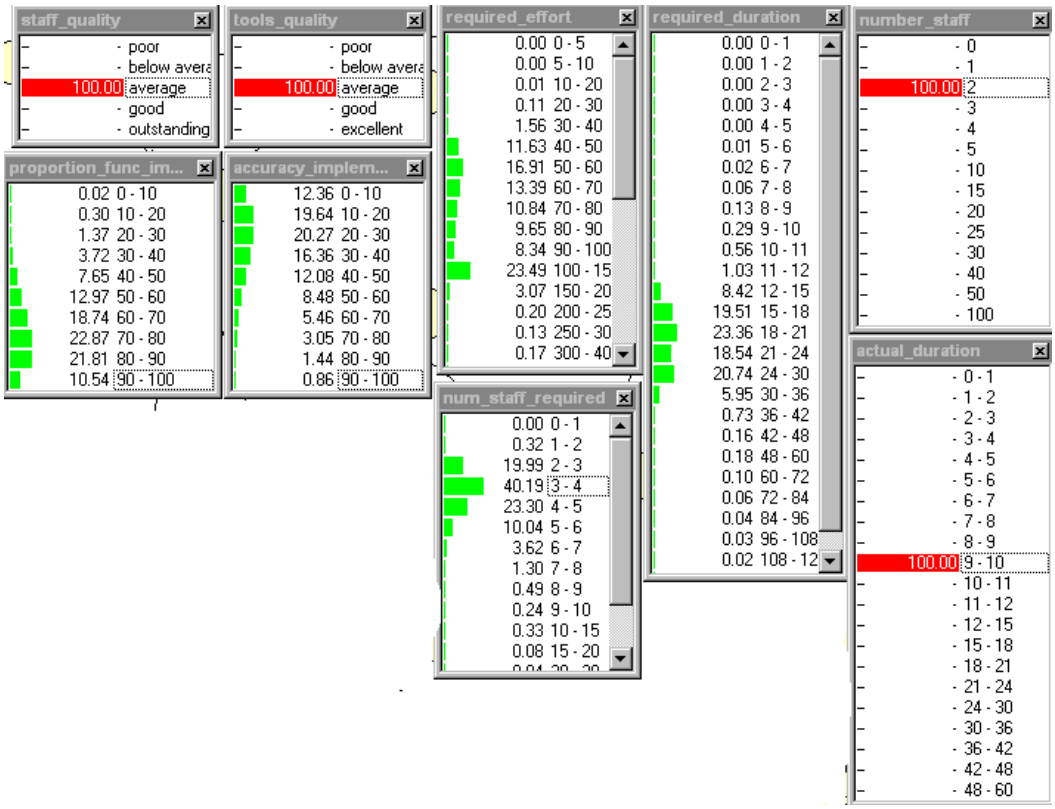

Figure 20 Actual resource "quality" information added, requirements on accuracy dropped

6 Conclusions

Classical regression-based approaches to modelling and predicting software defects and resources are inadequate and often flawed. They do not provide managers with decision support for risk assessment and hence do not satisfy the most important objectives of software metrics.

We have shown that causal models using BBNs have many advantages over the classical approaches. The benefits of using BBNs include:

- explicit modelling of 'ignorance' and uncertainty in estimates, as well as cause-effect relationships
- enables us to combine diverse types of information
- makes explicit those assumptions that were previously hidden - hence adds visibility and auditability to the decision making process
- intuitive graphical format makes it easier to understand chains of complex and seemingly contradictory reasoning
- ability to forecast with missing data.
- use of 'what-if?' analysis and forecasting of effects of process changes;
- use of subjectively or objectively derived probability distributions;
- rigorous, mathematical semantics for the model
- no need to do any of the complex Bayesian calculations, since tools like Hugin do this

The BBN models can be thought of as metrics-based risk management decision-support tools that build on the relatively simple metrics that are already being collected. These tools combine different aspects of software development and testing and enable managers to make many kinds of predictions, assessments and trade-offs during the software life-cycle, without any major new metrics overheads. The approach does, however, force managers to make explicit those assumptions which were previously hidden in their decision-making process. We regard this as a benefit rather than a drawback. The BBN approach is actually being used on real projects and is receiving highly favourable reviews. We believe it is an important way forward for metrics research.

Clearly the ability to use BBNs for accurate prediction still depends on some stability and maturity of the development processes. Organisations that do not collect the basic metrics data, or do not follow defined life-cycles, will not be able to apply such models effectively.

7 Acknowledgements

The work was supported, in part, by the EPSRC-funded project IMPRESS and used both the IMPRESS tool and the SERENE tool.

8 References

Abdel-Hamid TK, "The slippery path to productivity improvement", IEEE Software, 13(4), 43-52, 1996.

Adams E , "Optimizing preventive service of software products", IBM Research Journal, 28(1), 2-14, 1984.

Agena Ltd, "Bayesian Belief Nets", http://www.agena.co.uk/bbn_article/bbns.html

Akiyama F, "An example of software system debugging", Inf Processing 71, 353-379, 1971.

Albrecht A.J, Measuring Application Development, Proceedings of IBM Applications Development joint SHARE/GUIDE symposium. Monterey CA, pp 83-92, 1979.

Basili VR and Perricone BT, "Software Errors and Complexity: An Empirical Investigation", *Communications of the ACM*, Vol. 27, No.1, pp.42-52, 1984.

Boehm BW, Software Engineering Economics, Prentice-Hall, New York, 1981.

Compton T, and Withrow C. "Prediction and control of Ada software defects", *J. Systems and Software*, Vol. 12, 199-207, 1990.

Courtois P-J, Fenton NE, Littlewood B, Neil M, Strigini L, Wright D, Examination of bayesian belief network for safety assessment of nuclear computer-based systems, DeVa ESPRIT Project 20072,

3rd Year Deliverable, 1998.

Delic KA, Mazzanti F and Strigini L, Formalising a software safety case via belief networks, Proc DCCA-6, 6th IFIP International Working Conf on Dependable Computing for critical Appliations, Garmisch-Partenkirchen, Germany, March, 1997.

Fenton NE and Neil M, A Critique of Software Defect Prediction Models, IEEE Transactions on Software Engineering, to appear, 1999.

Fenton NE and Ohlsson N, Quantitative Analysis of Faults and Failures in a Complex Software System, IEEE Transactions on Software Engineering, to appear, 1999.

Fenton NE and Pfleeger SL, Software Metrics: A Rigorous and Practical Approach (2nd Edition), International Thomson Computer Press, 1996.

Fenton NE, Littlewood B, Neil M, Strigini L, Sutcliffe A, Wright D, Assessing Dependability of Safety Critical Systems using Diverse Evidence, IEE Proceedings Software Engineering, 145(1), 35-39, 1998.

Gaffney JR, "Estimating the Number of Faults in Code", *IEEE Trans. Software Eng.*, Vol.10, No.4, 1984.

Gilb T, Software Metrics, Chartwell-Bratt, 1976.

Halstead M, Elements of Software Science, North Holland, , 1977.

Hugin A/S: www.hugin.dk, Hugin Expert A/S, P.O. Box 8201 DK-9220 Aalborg, Denmark, 1998

Jensen FV, An Introduction to Bayesian Networks, UCL Press, 1996.

Jones C, "Applied Software Measurement", McGraw Hill, 1991.

Kitchenham BA, Using function points for software cost estimation, in 'Software Quality Assurance and Measurement' (Eds Fenton NE, Whitty RW, Iizuka Y), International Thomson Computer Press, 266-280, 1995.

Lauritzen SL and Spiegelhalter DJ, "Local Computations with Probabilities on Graphical Structures and their Application to Expert Systems (with discussion)". *J. R. Statis. Soc. Series B*, 50, No 2, pp.157-224, 1988.

Lewis NDC, Fenton N, Neil M, Uncertainty, software quality and statistical process control, Software Quality J, submitted, 1998.

Lipow M, "Number of Faults per Line of Code", *IEEE Trans. Software Eng.*, Vol. 8, No.4, 437-439, 1982.

McCabe T, A Software Complexity Measure, IEEE Trans. Software Engineering SE-2(4), 308-320, 1976.

Munson JC, and Khoshgoftaar TM, The detection of fault-prone programs, IEEE Transactions on Software Engineering,18(5), 423-433, 1992.

Putnam LH, A general empirical solution to the macro software sizing and estimating problem, IEEE Trans Soft Eng SE-4(4), 1978, 345-361, 1978.

SERENE consortium, SERENE (SafEty and Risk Evaluation using bayesian Nets): Method Manual, ESPRIT Project 22187, http://www.csr.city.ac.uk/people/norman.fenton/serene.htm, 1999.

Shen VY, Yu T, Thebaut SM, and Paulsen LR, "Identifying error-prone software - an empirical study", *IEEE Trans. Software Eng.*, Vol. 11, No.4, pp. 317-323, 1985.

Strigini L and Fenton NE, Rigorously assessing software reliability and safety, Proc Product Assurance Symposium and Software Product Assurance Workshop, 19-21 March 1996, ESA SP-377, May, 1996.

Symons, CR, Software Sizing & Estimating: Mark II Function point Analysis, John Wiley, 1991.

TRACS (Transport Reliability Assessment & Calculation System): Overview, DERA project E20262, http://www.agena.co.uk/tracs/index.html, 1999