Bayesian Networks for Software Process Control

Norman Fenton[1], Martin Neil[1], Paul Krause[2] and Rajat Mishra[3]
[1]Queen Mary, University of London and Agena Ltd.
London UK

[2]University of Surrey
Guildford UK (formerly Philips Research, Redhill)

[3]Philips Software Centre
Bangalore India

27 September 2005

Abstract - !!!Rewrite!!!

Although a number of approaches have been taken to quality prediction for software, none have achieved widespread applicability. Our aim here is to produce a single method to combine the diverse forms of, often causal, evidence available in software development in a more natural and efficient way than done previously. We use graphical probability models (also known as Bayesian Belief Networks) as the appropriate formalism for representing this evidence. We can use the subjective judgements of experienced project managers to build the probability models and use these models to produce forecasts about the software quality throughout the development life cycle. Moreover, the causal or influence structure of the model more naturally mirrors the real world sequence of events and relations than can be achieved with other formalisms.

We will introduce a general purpose tool, AgenaRisk, which is the result of an extended research programme involving Philips and a number of other partners. After introducing the tool we will describe the evaluation of one of the defect prediction models it supports. This was evaluated at PSC, Bangalore with very encouraging results.

1 Introduction

Despite several decades of intensive research, the problem of defining and measuring software reliability is still far from resolution. Let us start with a review of a statement in [Pham, 2000] that was used in the introduction to some foundational material on statistical reliability models:

"Consider a new and successfully tested system that operates well when put into service at time t = 0. The system becomes less likely to remain successful as the time interval increases. The probability of success for an infinite time interval, of course, is zero".

Leaving aside the meaning of "successfully tested" for the moment, this statement is still worth reflecting on. Assuming a single software installation, then no properties of the software product change, of course. Making a second assumption of the software being installed on perfectly reliable hardware, then the software will actually continue to execute without failure for so long as the usage patterns remain within the boundaries of the successful test scenarios (provided all faults discovered in the testing phase have been repaired). The reason for the apparent reduction in the probability of success is that as time progresses, then the likelihood of users exploring previously unexplored variable ranges, decision outcomes or program paths also increases. Hence, the likelihood of failure is a function of the likelihood that a usage scenario will move outside of the execution space that has been explored during testing, and the likelihood of hitting a fault if a previously unsampled region of the execution space is visited.

Note also that if known faults do remain after testing, we have certainty that failure will arise if the respective regions of the execution space are visited. However, the reliability of the software may be "acceptable" if the likelihood of a user visiting this region of the execution space is sufficiently low (and/or the impact of the resulting failure is sufficiently small).

In hardware reliability, failure likelihood is primarily a function of time as a result of the physical characteristics of a product changing from optimum through wear and ageing. In contrast, the failure likelihood of software is a function of the number of distinct internal operations executed; if only a limited

set of fault free operations are performed repeatedly and indefinitely, no failures will arise. The temporal element enters only because the likelihood of a given user moving outside the "safe" usage patterns increases as their needs and experience with the product matures over time.

We all know this. After comprehensive operational testing of a complex medical imaging system has been performed, for example, best practise is then to subject it to exploratory testing in house by testers familiar with good and bad clinical working practice and "the sort of problems that have arisen in the past" ("alpha-testing"). Then there will be a limited release of the product to customers where the product can be carefully monitored for failures ("beta-testing"). Only when confidence has been gained that both expected and unexpected usage patterns have been successfully covers and all critical faults corrected, will a full release take place. Yet still statistical software reliability models are predominantly designed to fit past failure data in terms of mean time to failure or failure intensity, rather than to collate evidence to predict the likelihood of a new user, or an unforeseen usage pattern, exercising a new fault condition.

Instead of focussing on execution time based models of reliability, in this paper we will focus on predicting more general measures of user satisfaction, and on software defect prediction. In particular, we believe that focusing on the prediction of residual defects is a first step in deconstructing the notion of software reliability on order to explicitly identify all the factors which impact on user-perceived reliability.

We will report in this paper on the use of Bayesian Networks to model the causal influences on software design quality during software development. Currently this has resulted in two distinct kinds of network models. In the first, we use a single network to model the overall development cycle of a product. This provides an extension of industry standard effort prediction models to enable us to balance functionality and quality delivered against resourcing and time-to-market constraints. This kind of model has the potential to place high-level project and portfolio management decisions on a quantitative basis. The second kind of model uses a sequence of Bayesian networks to model the whole project lifecycle in detail. These models are intended for more detailed defect predictions.

Extended evaluations of both kinds of models have been performed at the Philips Software Centre, Bangalore. The outcomes of these evaluations have been very encouraging and are reported on in detail in this paper.

2 The problems with software defect prediction

Fenton and Neil [1999] provide a detailed critique of software defect prediction models. The essential problem is the oversimplification that is generally associated with the use of simple regression models. Typically, the search is for a simple relationship between some predictor and the number of defects delivered. Size or complexity measures are often used as such predictors. The result is a naïve model that could be represented by the graph of Figure 2.1.

The difficulty is that whilst such a model can be used to *explain* a data set obtained in a specific context,

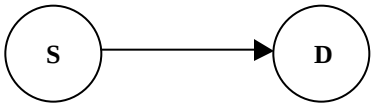


Figure 2.1: Graphical representation of a naïve regression model between some predictor S (typically a size measure), and the number of software defects D.

none has so far been subject to the form of controlled statistical experimentation needed to establish a *causal* relationship. Indeed, the analysis of Fenton and Neil suggests that these models fail to include all the causal or explanatory variables needed in order to make the models generalisable. Further strong empirical support for these arguments is demonstrated in [Fenton and Ohlsson, 2000].

As an example, in investigating the relationship between two variables such as S and D in Figure 2.1, one would at least wish to differentiate between a direct causal relationship and the influence of some common cause as a "hidden variable". For example, we might hypothesise "Problem Complexity" (PC) as a common cause for our two variables S and D, Figure 2.2.

The model of Figure 2.1 can simulate the model of Figure 2.2 under certain circumstances. However, the latter has greater explanatory power, and can lead to quite a different interpretation of a set of data. One

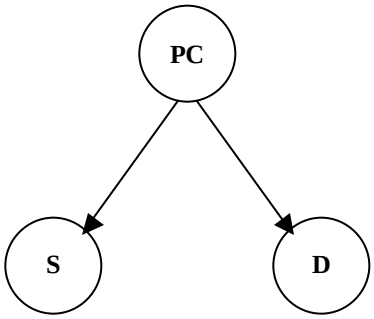


Figure 2.2: The influence of S on D is now mediated through a common cause PS. This model can behave in the same way as that of Figure 2.1, but only in certain specific circumstances.

could take "Smoking" and "Higher Grades" at high school as an analogy. Just looking at the covariance between the two variables, we might see a correlation between smoking and achieving higher grades. However, if "Age" is then included in the model, we could have a very different interpretation of the same data. As a student's age increases, so does the likelihood of their smoking. As they mature, their grades also typically improve. The covariance is explained. However, for any fixed age group, smokers may achieve lower grades than non-smokers.

We believe that the relationships between product and process attributes and numbers of defects are too complex to admit straightforward curve fitting models. In predicting defects discovered in a particular project, we would certainly want to add additional variables to the model of Figure 2.2. For example, the number of defects discovered will depend on the effectiveness with which the software is tested. It may also depend on the level of detail of the specifications from which the test cases are derived, the care with which requirements have been managed during product development, and so on. We believe that graphical probabilistic models are the best candidate for situations with such a rich causal structure.

3 Introduction to probabilistic models

3.1 Bayes' theorem and graphical models

Probability is a dynamic theory; it provides a mechanism for coherently revising the probabilities of events as evidence becomes available. Bayes' theorem is a fundamental component of the dynamic aspects. We will provide a reminder of Bayes' theorem in this section for completeness, and then revise the basics of graphical probabilistic models. This provides the context for introducing the additional features that needed to be developed in order to implement a tool that provided extensive support for customising the graphical models.

We write p(A | B) to represent the probability of some event (an hypothesis) conditional on the occurrence of some event B (evidence). If we are counting sample events from some universe Ω, then we are interested in the fraction of events B for which A is also true. In effect we are focusing attention from the universe Ω to a restricted subset in which B holds. From this it should be clear that (with the comma denoting conjunction of events):

$$p(A \mid B) = \frac{p(A, B)}{p(B)}$$

This is the simplest form of Bayes' rule. However, it is more usually rewritten in a form that tells us how to obtain a posterior probability in a hypothesis A after observation of some evidence B, given the *prior* probability in A and the likelihood of observing B were A to be the case:

$$p(A \mid B) = \frac{p(B \mid A)p(A)}{p(B)}$$

This theorem is of immense practical importance. It means that we can reason both in a forward direction from causes to effects, and in a reverse direction (via Bayes' rule) from effects to possible causes. That is, both deductive and abductive modes of reasoning are possible.

However, two significant problems need to be addressed. Although in principle we can use generalisations of Bayes' rule to update probability distributions over sets of variables, in practice:

1) Eliciting probability distributions over sets of variables is a major problem. For example, suppose we had a problem describable by seven variables each with two possible states. Then we will need to elicit (2^7-1) distinct values in order to be able to define the probability distribution completely. As can be seen, the problem of knowledge elicitation is intractable in the general case.

2) The computations required to update a probability distribution over a set of variables are similarly intractable in the general case.

Up until the late 1980's, these two problems were major obstacles to the rigorous use of probabilistic methods in computer based reasoning models. However, work initiated by Lauritzen and Spiegelhalter [1988] and Pearl [1988] provided a resolution to these problems for a wide class of problems. This work related the independence conditions described in graphical models to factorisations of the joint distributions over sets of variables. We have already seen some simple examples of such models in the previous section. In probabilistic terms, two variables X and Y are independent if $p(X,Y) = p(X)p(Y)$ – the probability distribution over the two variables factorises into two independent distributions. This is expressed in a graphic by the *absence* of a direct arrow expressing influence between the two variables.

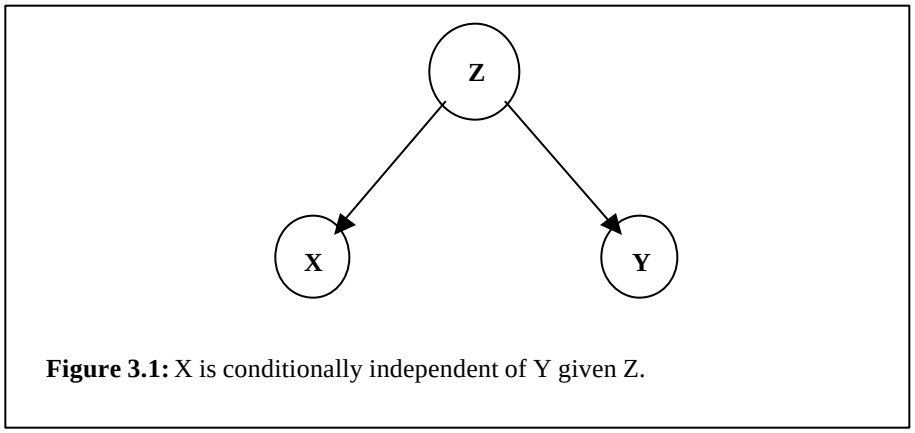


Figure 3.1: X is conditionally independent of Y given Z.

We could introduce a third variable Z, say, and state that "X is conditionally independent of Y given Z". This is expressed graphically in Figure 3.3. An expression of this in terms of probability distributions is:

$$p(X,Y \mid Z) = p(X \mid Z)p(Y \mid Z)$$

A significant feature of the graphical structure of Figure 3.3 is that we can now decompose the joint probability distribution for the variables X, Y and Z into the product of terms involving at most two variables:

$$p(X,Y,Z) = p(X \mid Z)p(Y \mid Z)p(Z)$$

In a similar way, we can decompose the joint probability distribution for the variables associated with the nodes DD, TE and SQ of Figure 4.2 as

$$p(DD, TE, SQ) = p(DD \mid TE,SQ)p(TE)p(SQ)$$

This gives us a series of example cases where a graph has admitted a simple factorisation of the corresponding joint probability distribution. If the graph is directed (the arrows all have an associated direction) and there are no cycles in the graph, then this property is a general one. Such graphs are called Directed Acyclic Graphs (DAGs). Using a slightly imprecise notation for simplicity, we have [Lauritzen and Spiegelhalter, 1988]:

Proposition

Let $U = \{X_1, X_2, \ldots, X_n\}$ have an associated DAG G. Then the joint probability distribution p(U) admits a direct factorisation:

$$p(U) = \prod_{i=1}^{n} p(X_i \mid pa(X_i))$$

Here $pa(X_i)$ denotes a value assignment to the parents of X_i. (If an arrow in a graph is directed from A to B, then A is a parent node and B a child node).

The net result is that the probability distribution for a large set of variables may be represented by a product of the conditional probability relationships between small clusters of semantically related propositions. Now, instead of needing to elicit a joint probability distribution over a set of complex events, the problem is broken down into the assessment of these conditional probabilities as parameters of the graphical representation. This resulted in a significant advance in the applicability of probabilistic networks to real-world problems. However, even for a single node, assessing the conditional probability table can be a daunting prospect (e.g. 625 values for a node with three parents, where all nodes have five possible states). In addition, our earlier experience with the use of probabilistic networks for process modelling quickly threw up requirements for easy customisation of existing networks (ideally by non-expert users), modularisation of networks to enable new processes to be modelled by composing existing models, and dynamic discretisaton of nodes with continuous variables to support large ranges of possible values.

Drawing on this work in various commercial projects with Agena, Fenton and Neil have built BN-based applications that have proved the technology is both viable and effective. Several of these applications have been related to systems or software assessment. Especially significant was the TRACS tool [18] to assess vehicle reliability for QinetiQ (on behalf of the MOD) and the AID tool [20] to predict software defects in consumer electronic products for Philips. Much of the modelling work described here has been done as part of the MODIST project [9], which extends the ideas in AID. The toolset implementation has been based on Agena's AgenaRisk technology that was extended to incorporate recent developments in building large-scale BNs that was undertaken in the SCULLY, SIMP and SCORE projects [11].

Let us take a look at an example model to illustrate some of these points. Figure 3.2 illustrates a simple model that can be taken as the foundation for some of the process models that are discussed later in this paper.

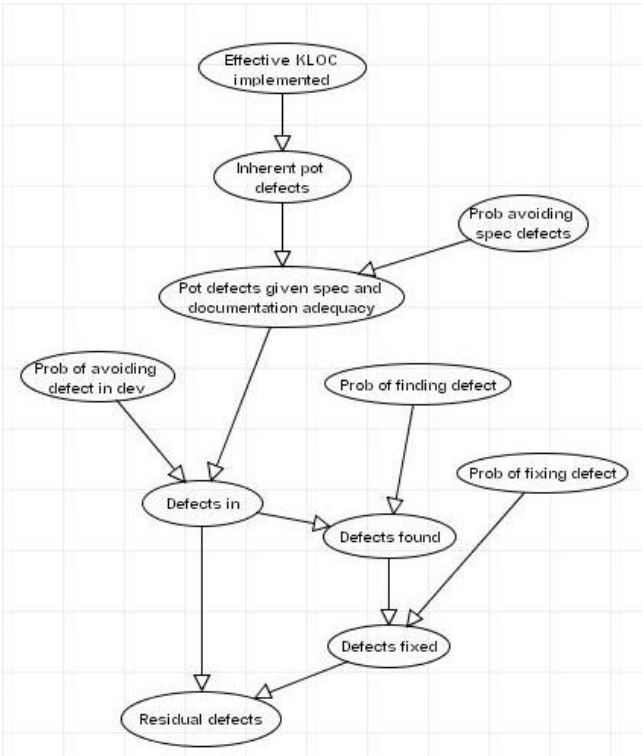


Figure 3.2 Foundational model for defect prediction

The BN of Figure 3.2 forms a causal model of the process of inserting, finding and fixing software defects. The variable 'effective KLOC implemented' represents the complexity-adjusted size of the functionality implemented; as the amount of functionality increases the number of potential defects rises.

The 'probability of avoiding defect in development' determines 'defects in' given total potential defects. This number represents the number of defects (before testing) that are in the new code that has been implemented.

However, inserted defects may be found and fixed: the residual defects are those remaining after testing. Variables representing a number of defects take a value in a numeric range, discretised into numeric interval.

There is a probability table for each node, specifying how the probability of each state of the variable depends on the states of its parents. Some of these are deterministic: for example the 'Residual defects' is simply the numerical difference between the 'Defects in' and the 'Defects fixed'. In other cases, we can use standard statistical functions: for example the process of finding defects is modelled as a sequence of independent experiments, one for each defect present, using the 'Probability of finding a defect' as a characteristic of the testing process:

Defects found = B(Defects inserted, Prob finding a defect)

where *B(n,p)* is the Binomial distribution for *n* trials with probability *p*. For variables without parents the table just contains the prior probabilities of each state.

The BN represents the complete joint probability distribution – assigning a probability to each combination of states of all the variables – but in a factored form, greatly reducing the space needed. When the states of some variables are known, the joint probability distribution can be recalculated conditioned on this 'evidence' and the updated marginal probability distribution over the states of each variable can be observed.

The quality of the development and testing processes is represented in the BN of Figure 3.2 by four variables discretised over the 0 to 1 interval:

- probability of avoiding specification defects
- probability of avoiding defects in development
- probability of finding defects
- probability of fixing defects.

Two features of AgenaRisk are especially critical for building this model:

- Large tables can be handled efficiently. For example, in the default model here the number of defects may range from 0 to 3000, in intervals of varying size.
- Probability tables are generated from numerical and statistical expressions by simulation. The expression given above using the binomial distribution is not only the conceptual model but also how the model is specified.

Wherever possible, the tables were generated using statistical data; either published data, or data obtained from our own records. If such data is not available, however, expert judgement can be used to augment the probability tables. In the latter case, careful elicitation procedures can result in trustworthy data, although usually with a loss in precision over hard statistical data. Whether statistical data or expert judgement is used, our experience has been that the tool support for generating tables from expressions greatly facilitates the model development phase, and the construction of tables by entering individual values into each cell was very rarely performed.

3.2 Varying the lifecycle

When we describe defects being inserted in 'implementation' and removed in 'testing' we are referring to the activities that make up the software development lifecycle. We need to fit a decision support system to the lifecycle being used but practical lifecycles vary greatly. In this section, we describe how this can be achieved without having to build a bespoke BN for every different lifecycle. The solution has two steps: the idea of a lifecycle 'phase' modelled by a BN and a method of linking separate phase models into a model for an entire lifecycle.

We model a development lifecycle as made up from 'phases', but a phase is not a fixed development process as in the traditional waterfall lifecycle. Instead, a phase can consist of any number and

combination of such development processes. For example, in the 'incremental delivery' approach the phases could correspond to the code increments; each phase then includes all the development processes: specification, design, coding and testing. Even in a traditional waterfall lifecycle it is likely that a phase includes more than one process with, for example, the testing phases involving some new design and coding work.

The incremental and waterfall models are just two ends of a continuum. To cover all parts of this continuum, we consider all phases to include one or more of the following development activities:

- Specification/documentation: This covers any activity whose objective is to understand or describe some existing or proposed functionality. It includes: requirements gathering writing, reviewing, or changing any documentation (other than comments in code).
- Development (or more simply coding): This covers any activity that starts with some predefined requirements (however vague) and ends with executable code.
- Testing and rework: This covers any activity that involves executing code in such a way that defects are found and noted; it also includes fixing known defects.

The phase BN includes all these activities, allowing the extent of each activity in any actual phase to be adjusted. In the most general case, a software project will consist of a combination of these phases. In order to be able to model an arbitrary lifecycle, we need to be able to link together multiple instances of a BN.

Whatever the development lifecycle, the main objective is: given information about current and past phases we would like to be able to predict attributes of quality for future phases. We therefore think of the set of phases as a time series that defines the project overall. This is readily expressed as a Dynamic Bayesian Network (DBN) [2]. A DBN allows time-indexed variables: in each time frame one of the parents of a time-indexed variable is the variable from the previous time frame. Figure 3.3 show how this is applied when the quality attribute is the number of residual defects.

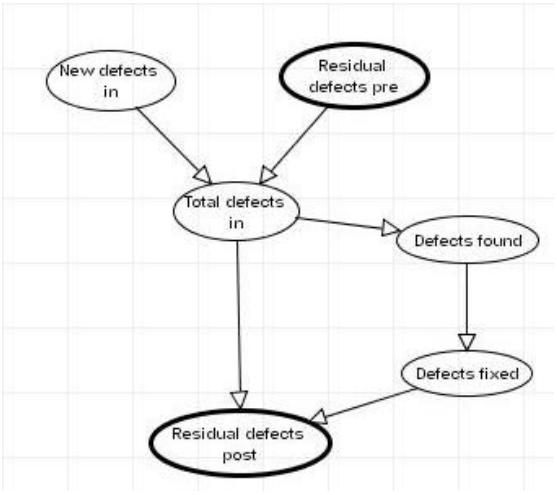


Figure 3.3 A Dynamic BN Modelling a Software Lifecycle

The dynamic variable is shown with a bold boundary. We construct the DBN with two nodes for each time-indexed variable: the value in the previous time frame is the 'input' node (here 'Residual defects pre') and it has no parents in the net. The node representing the value in this time frame is called the 'output node' (here 'Residual defects post'). Note that the variable for the current time frame 'Residual defects post' depends on the one for the previous time frame, but as an ancestor rather than as a parent since it is clearer to represent the model with the intermediate variable 'Total defects in'.

As well as defects, we also model the documentation quality as a time-varying quality attribute. Recall that documentation includes specification, which even in iterative developments is often prepared in one phase and implemented in a later phase. We consider specification errors as defects so a phase in which documentation is the main activity may lead to an important incremental change in documentation quality that is passed on to the next phase.

4 A Bayesian Network for Project Management

The preceding sections have outlined the basic theory that lies behind our model building approach. This foundational work has underpinned the development of a flexible tool for developing and using probabilistic models in a range of problem areas. The resulting tool is now marketed as AgenaRisk (see: www.agena.co.uk), and the remainder of this paper will focus on the evaluation of that tool.

The AgenaRisk toolset supports an extensive range of customisable assessment models. These models capture the causal influences on the quality of a software project. Rather than focus on the underlying Bayesian Networks, however, we will focus on usage of the models via a questionnaire based User Interface. This will keep the focus on the actual benefits in use of the AgenaRisk toolset. Further background on the use of Bayesian Networks for software process modelling can be found in [Fenton et al, 2002].

Figure 4.1 is an image of the questionnaire view of the assessment model for a complete software project.

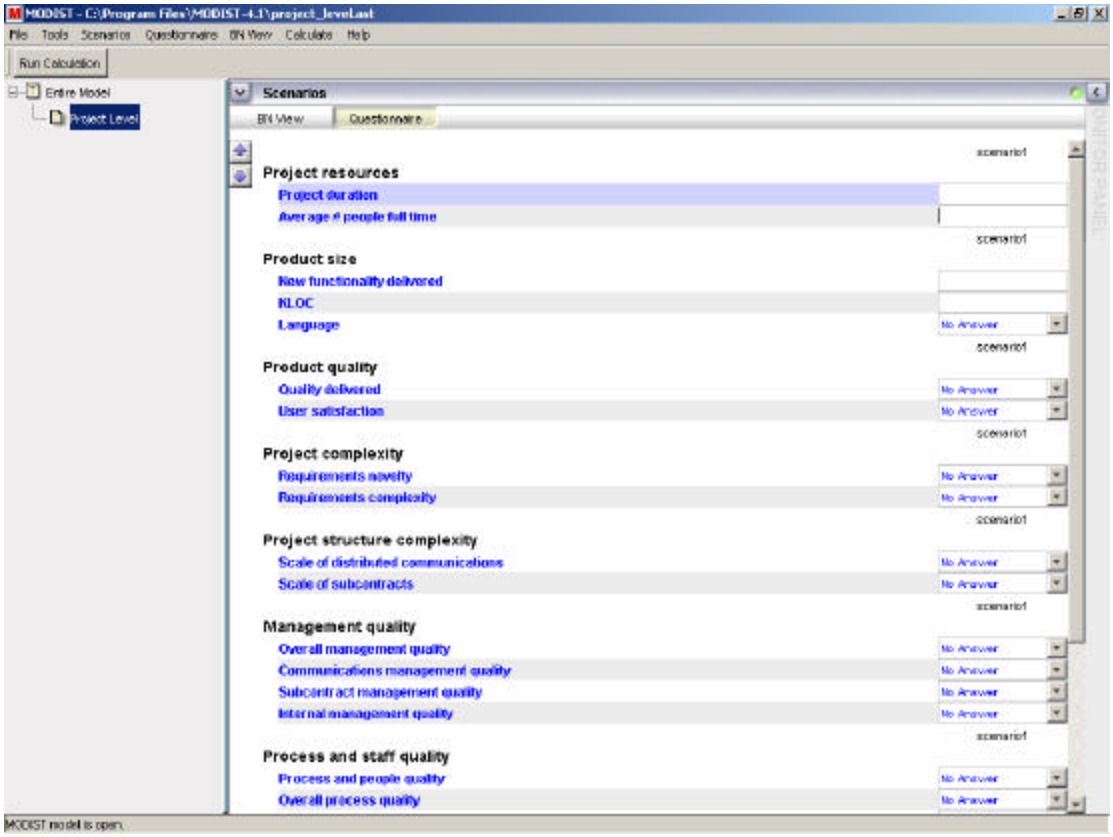
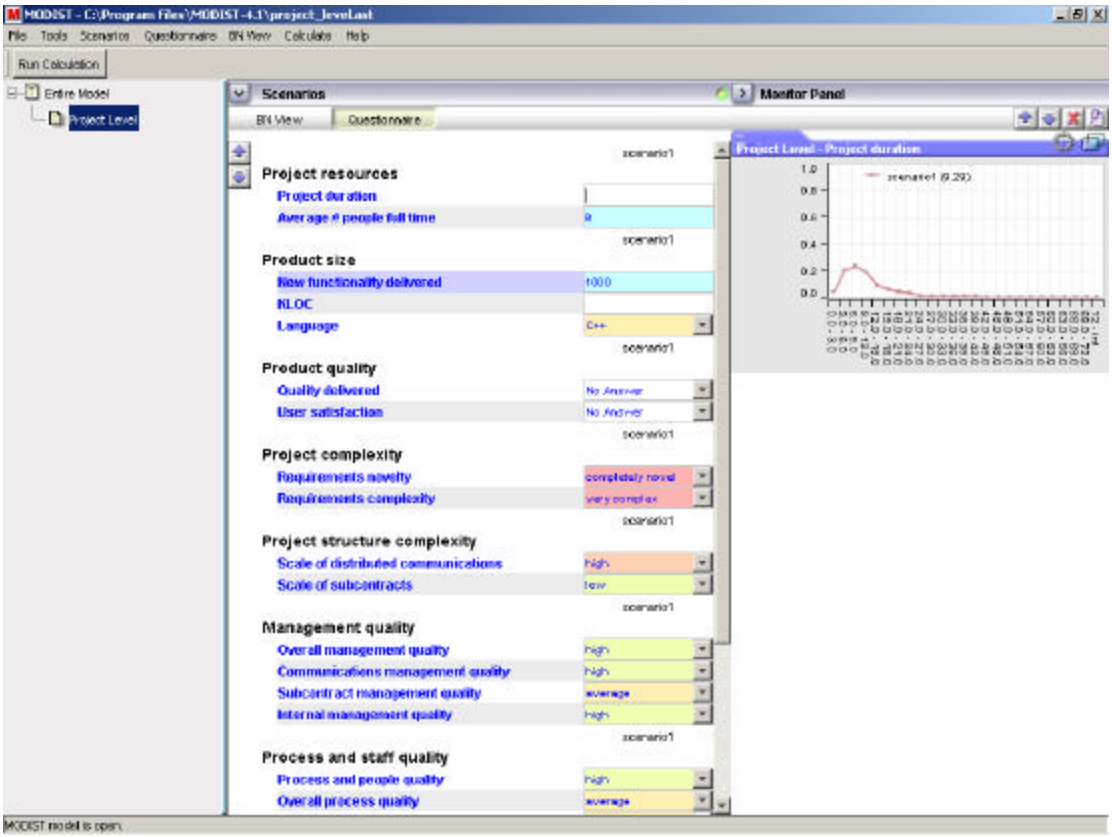


Figure 4.1: AgenaRisk in questionnaire mode

The project level BN is a very general-purpose quality and risk assessment model for distributed software development projects. By viewing the distributed project at this level, the model enables us to predict different aspects of resources and quality while monitoring and mitigating different types of risk.

A very significant feature of the use of BNs is that they are extremely flexible in the way that they can be used. As we see, the interface provides a range of questions that could be answered by a user of the tool. Depending on precisely what we would like to achieve, we may answer specific sets of questions and view the predicted answers for the remaining questions. For example, suppose we wish to estimate how long a certain project will take. We enter details of the project, in terms of complexity, size and a range of aspects of the development process that is proposed. We then display the monitor to display the prediction for "Project duration" – the specific attribute we are interested in. Once all the information that is available has been entered, then the assessment calculations can be performed. The model will then display a prediction for the expected duration of the project (Figure 4.2):

Figure 4.2: Entry of Project Data



We can see that the tool is making a prediction of the project duration with a median of 9.3 months. Note that the monitor on the right hand side is displaying the prediction as a distribution over a range of possible values. This is a deliberate design choice. Any form of prediction in software development has an inherent uncertainty associated with it, and we believe it is important to make this uncertainty explicit.

Now, having made a prediction for the duration of this specific project, we can commit to an actual value close to the median value displayed with some confidence that this will actually be achievable. We can begin costing out the project for our customer. Suppose we quote a delivery time of 9 months with the 8 full time equivalent people that have been allocated to the project. Given that this duration is consistent with the AgenaRisk predictions, we can realistically cost the project on this basis. That is fine, but how confident are we that the delivered product will meet the user's expectations?

Again, AgenaRisk can help us. We commit to the 9month duration by entering this value in the questionnaire. Note how AgenaRisk allows us to switch between qualitative and quantitative values for any attribute. Then we open up the monitors for "Quality Delivered" and "User Satisfaction". Typically, these monitors will have been scaled to provide respective measures that conform to an organisation's specific measures of success. However, for generality we will retain the default scaling. Interestingly, we see that although the expectation for Quality Delivered is Good, we only have a Satisfactory for User Satisfaction. Quality Delivered is essentially referring to the maturity of the proposed solution. AgenaRisk is raising no especial concerns about this. User Satis faction on the other hand, is referring to conformance of that solution to the User's expectations. Here, AgenaRisk is being less optimistic; the expectation is only for "average" satisfaction (Figure 4.3).

Notice that a change in one attribute, duration, has distinctly different effects on these two "result nodes". A major benefit of causal models is their ability to reflect the different strengths of the interdependencies that exist among all attributes of the process being modelled.

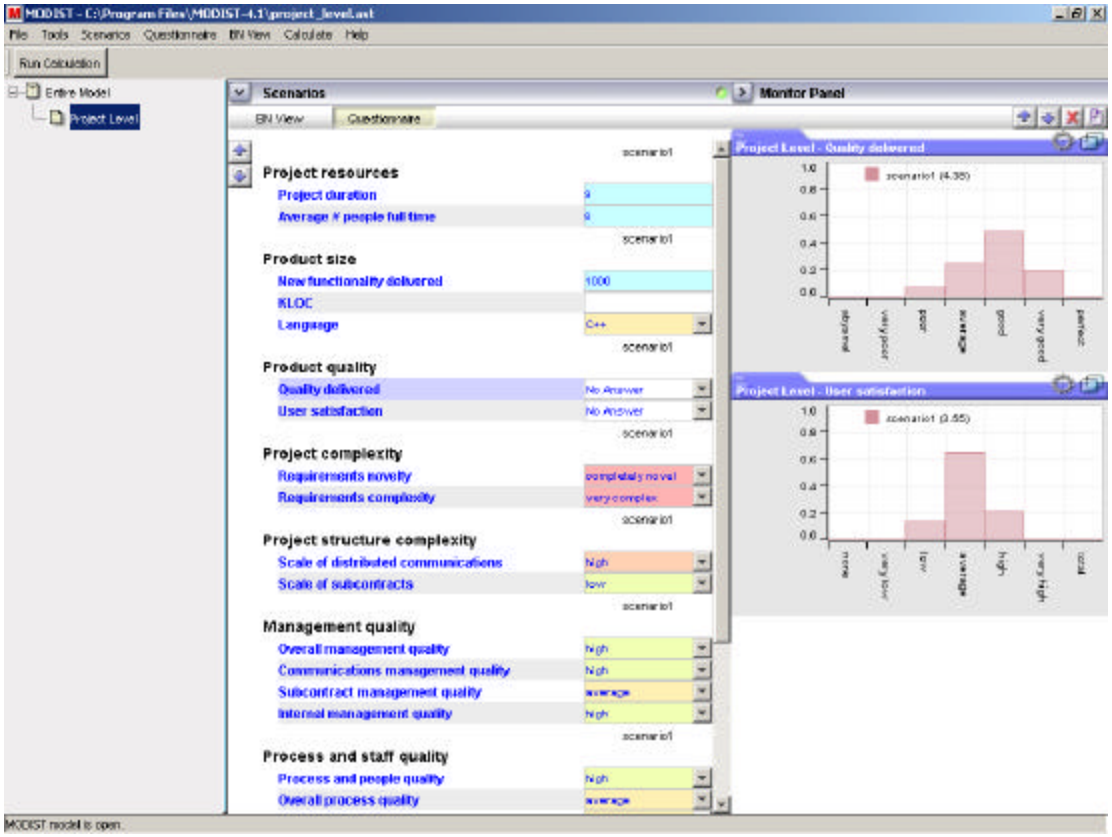


Figure 4.3: Display of Result Nodes

Why is this so, and what can we do to improve the situation? Bayesian networks can be run from cause to predicted effect (as here). However, they can also be run in a "reverse" direction, from effect to identify possible causes. So, one option would be to enter a value of very high or total for User Satisfaction and see what it tells us about the required values for all the other process attributes. This can be helpful, but tends to be somewhat unfocused – generally we will see recommendations for improvements across the whole development process.

We can obtain more focussed advice by exploring two or three alternative "what if" scenarios. In this particular case, we first note that Requirements novelty indicates that this is a "first of a kind" product development, and of some complexity. This may be responsible for the high risk that the project may not be fully satisfactory in meeting the User's needs. We cannot change these requirements, so what other project attributes could be used to mitigate this risk?

If we scroll down the questionnaire a little further (Figure 4.4), we see that there are a number of attributes associated with the product specification process that we could change. Suppose we were to involve the project stakeholders more closely, in order to produce more precise and clearer specifications? We revise the questionnaire and update the predictions. As we can see from Figure 4.4, User Satisfaction now has a significantly higher likelihood of being acceptable.

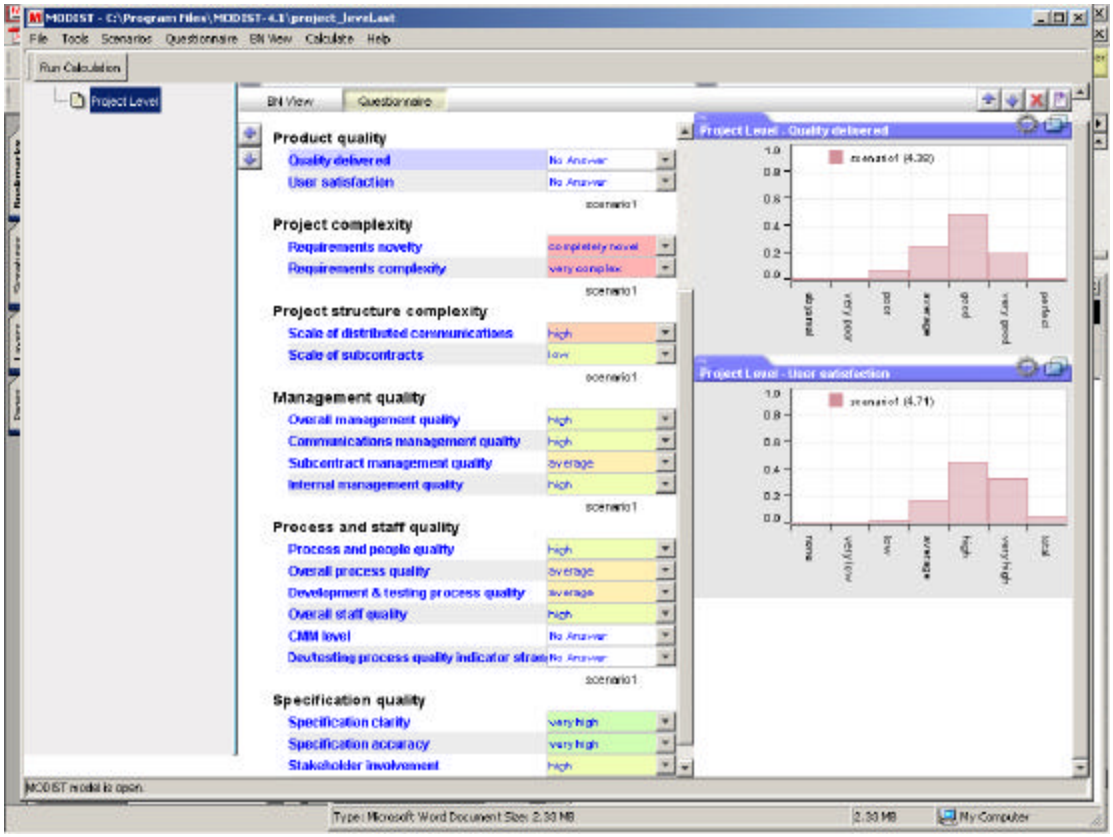


Figure 4.4: Process Improvements for higher User Satisfaction

Unfortunately, we are not yet finished. Our customer is not satisfied with the cost and delivery time. They feel we can deliver the product with fewer people, and in less time. So, what is the risk if we reduce the resourcing, and try and push the project through in a shorter time? Again, AgenaRisk can help us in making this explicit. We clone our first development scenario. Then in this new scenario, Scenario 2 of Figure 4.5, we revise the Project resourcing to 6 people with a delivery time of 6 months. Figure 4.5 shows the customer that we can do this, but there is a very significant risk to both delivered quality and potential user satisfaction.

We could offer the customer a revised contract, with lower cost and shorter delivery time. However, they would need to accept the risks that AgenaRisk has identified in terms of poorer quality and lowered expectations that the delivered solution will be fully satisfactory to that customer.

The display of multiple scenarios has been used here to illustrate how AgenaRisk can be used to identify a preferred way of executing a *single* project from a number of alternatives. However, the scenarios can also correspond to different projects, to enable a project manager to obtain an overview of the status of *multiple* projects. Hence, this approach has potential to provide support for portfolio management.

AgenaRisk has a range of assessment models that can be used to model the software development process in more detail than the Project Level network. However this example case illustrates the main benefits of using AgenaRisk. As can be seen, AgenaRisk provides an extremely dynamic and flexible way of assessing, monitoring and controlling software development projects.

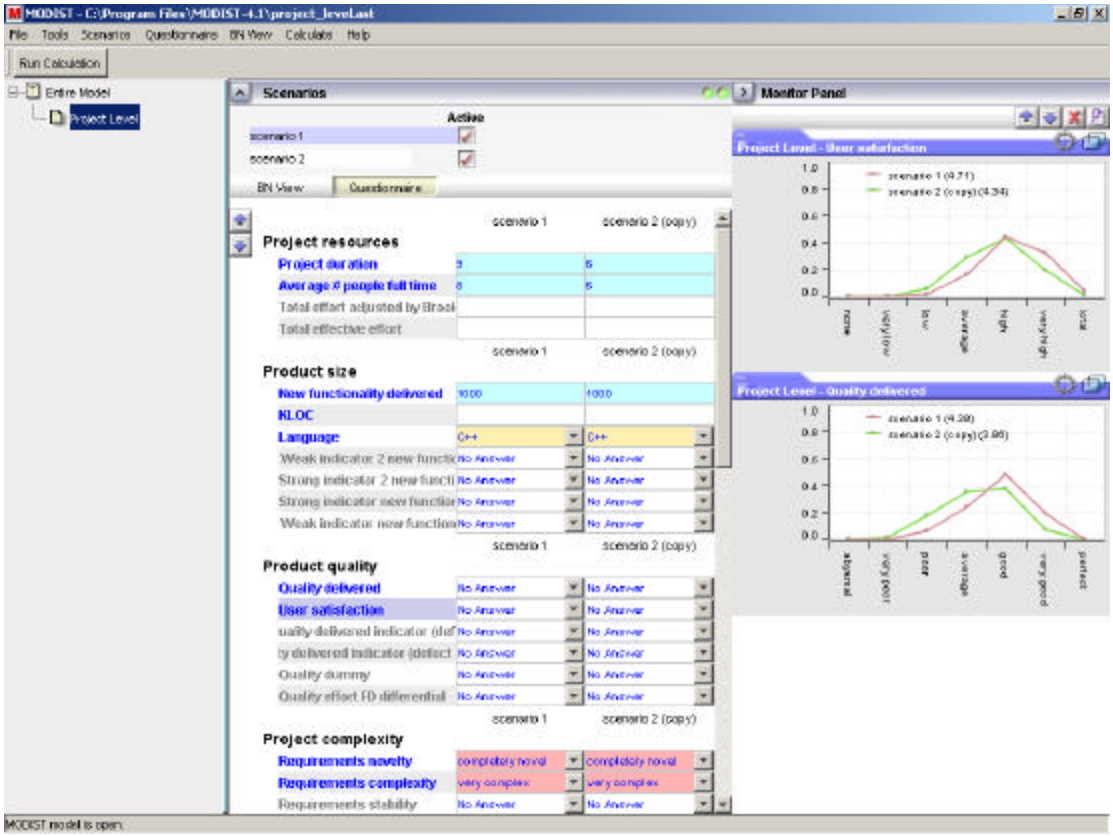


Figure 4.5: Display of Multiple Scenarios to Illustrate Potential Risks

4.1 Experience Report on use of the project level model

Although the project-level model has been demonstrated to and trialled by a range of potential users in the software industry, it has been hard to date to obtain sufficient data to perform a rigorous statistical evaluation. To an extent, just evaluating the tool by comparing predicted outcomes versus actual outcomes is missing the point of the tool. Instead, we recommend that the tool be used at project inception to provide an assessment of whether the quality goals are likely to be achieved given the novelty of the proposed product, the nature of the resources that are available and the planned time to market. If the indications from the model are that there is a high risk of failure to meet the quality goals, then the tool can be used to run a number of "what-if" scenarios to estimate the kind and level of corrective actions that are needed.

We have used the project-level network experimentally to monitor the progress of a high-level consumer electronics project. The first task is to calibrate one or both of the outcome nodes (User satisfaction and Quality delivered) in terms of measures that are used within the respective organisation. In our case, we used Field Call Rate, which is basically a measure of customer-raised issues normalised by the total number of products sold. Note that field calls may include customer issues that are due to usability issues and not necessarily the result of failures in the field in the strict sense of non-conformance to specified behaviour. Hence this can be a complex measure to predict, so to provide a more limited test of the network we restricted the evaluation to Field Calls that were directly traceable to software faults.

The first task was to calibrate the "Quality delivered" node in terms of Field Call rate. This was performed using data from past projects to provide a benchmark for "average" and then scaling for the high and low ends of the scale. We then entered the normal project data for a fresh-start product that was to be a platform for demonstrating a number of technological innovations. Not surprisingly perhaps, the initial prediction was for there to be a significant regression on the steady reduction in FCR that had been achieved for products over the last few years. Following this, execution of a range of what if scenarios indicated that a set of very tightly monitored process improvement actions needed to be followed in order to achieve the goal of maintaining significant reductions in FCR.

Given that AgenaRisk was a new tool to the organisation, it would be an act of hyperbole to say that these results were responsible for motivating the stringent controls that were placed on this development project. However, they did add weight to the debate over the importance of these, and we are pleased to say that an extremely successful outcome was achieved that was consistent with our best-case predictions.

Similar evaluations have been performed at Israel Aircraft Industries and QinetiQ, again with very good feedback from those involved. The tool is showing great value as a project management tool. The ability to handle multiple scenarios can also be used effectively for portfolio management where a single screen can be used to optimise the distribution of resources across a range of products.

5 The phase model BN

5.1 Background

The project risk model described in Section 4 enables managers to make decisions about a distributed system project by looking at it as a whole. Inevitably the variables are 'global' in the sense, for example, that we are concerned with the quality of the entire team (rather than specific teams and individuals) and the quality of the entire project (rather than specific modules/subsystems). Yet, if we want any kind of detailed defect prediction we need to work at the lower level of granularity. The so-called 'phase model' described in this section is the heart of our method for defect prediction.

The overriding objective of the phase model is to be able to predict defects and defect rates at different periods during a software development project based on information available at any stage of development and testing. For example, suppose our development process follows a traditional 'waterfall' life-cycle where we might have the following set of sequential phases:

- requirements capture
- specification
- design
- coding
- unit testing
- integration testing
- system testing
- acceptance testing
- operational use.

Then at the unit testing phase we could use information about defects found then (together with previous information from specification, design and coding) to predict defects at the system testing phase and later. We could also use such information to revise our beliefs about previous phases.

If we knew that all projects followed a fixed life-cycle then it would be possible to use a fixed BN model that captured such a life-cycle (indeed the AID tool, which was developed for Philips Consumer Electronics in the early stages of this research programme [Fenton *et al*, 2002], made precisely such an assumption). However, our method must enable us to build tailorable BNs that are relevant for projects with arbitrarily different or complex life-cycles. To enable us to do this (and to enable the models to reflect real software development practices) we have to think in terms of a software project as comprising an arbitrary number of phases that take place over time.

A phase is not necessarily a fixed or pre-defined 'life-cycle' process as in the traditional waterfall model. Instead, a phase can consist of any number and combination of such life-cycle processes. For example, in the 'incremental delivery' approach the phases could correspond to the code increments; inside each code increment a range of specification, design, coding and testing phases take place. Even in a traditional waterfall development it is highly unlikely that, for example, the testing phases would not involve some new design and coding work. The incremental and waterfall models are just two ends of a continuum. In fact, in practice, for many projects what determines the sequential phases is purely time: phase 1 might simply be 'the first month of the project', phase 2 the second month etc.

Irrespective of what actually defines a 'phase', the main objective remains: given information about current and past phases we would like to be able to predict attributes of quality for future phases. We therefore think of the set of phases as a time series that defines the project overall.

Attempting to model explicitly the details of all possible combinations of any software phases in any time period is computationally intractable even if we use the notion of extended BNs described in [Agena 2002b]. The core concept that will enable us to achieve our objective is the notion that any phase in a software project comprises one or more of the following activities:

- Specification/documentation: This covers any activity whose objective is to understand or describe some existing or proposed functionality. It includes:
    - requirements gathering
    - writing, reviewing, or changing any documentation (other than comments in code). Hence it includes specification documents, design documents and user manuals.
- Development (or more simply coding): This covers any activity that starts with some pre-defined requirements (however vague) and ends with executable code.
- Testing and rework: This covers any activity that involves executing code in such a way that defects are found and noted; it also includes fixing known defects.

Thus, in the most general case, in any phase our software project will consist of a combination of the above activities. We will provide now an overview of the "All Activities" phase net. This is for two reasons. Firstly, and perhaps most importantly, it is this network that was the subject of the evaluation activity at Philips Software Centre, Bangalore. Secondly, it does provide an overview of the modelling approach that has been followed in the development of all the different kinds of phase level nets. Although we will focus on a single network, do remember that the full generality of the approach can only be demonstrated through the use of multiple networks composed together to provide an accurate representation of the development process that is being modelled.

5.2 Overview of the "All Activities" Phase Level Network

To best understand the rationale behind this BN (which is based on experience with AID as well as years of practical software development experience) we consider the most general example of when the phase is concerned with developing and testing some part of a module/subsystem in a distributed project.

1. Since we are to build and test some part of a module we must be implementing some new functionality in this phase. The subnet ‘new functionality implemented' provides a measure of the size of this functionality.

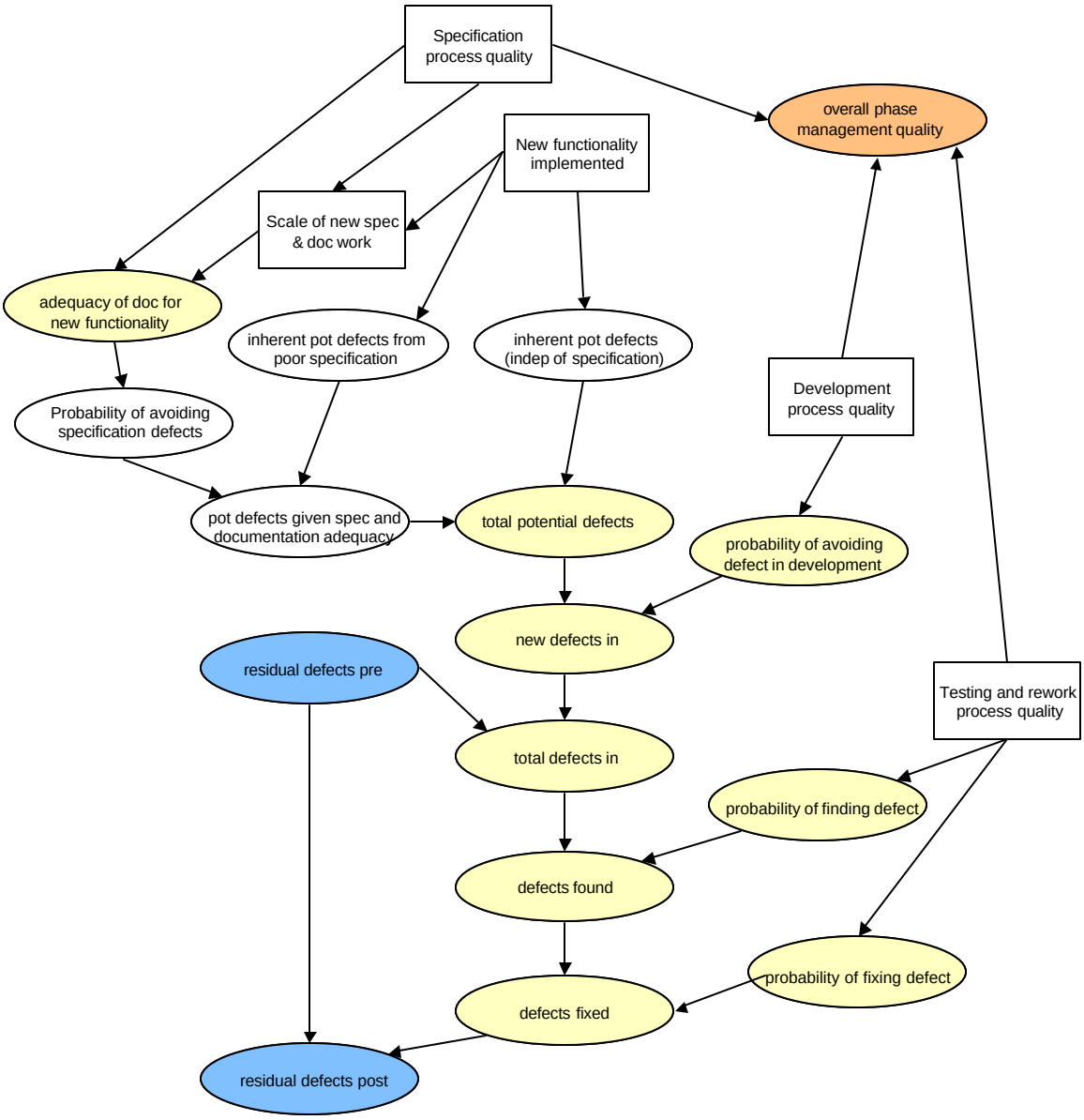


Figure 5.1: Schematic view of the whole phase net. Note that some of the detail is hidden inside the square boxes.

2. Before implementing any functionality there is assumed to be some specification of it. If we are lucky this specification will be a well-written document at the appropriate level of detail. However, in many cases it may be nothing more than a vague statement of requirements (which may not even be written down). Thus, generally there may be work that needs to be done on the specification. Hence we have a subnet concerned with measuring ‘scale of new specification and documentation work’. This will be dependent on variables in the subnet concerned with ‘specification process quality’.
3. The ‘scale of new specification and documentation work’ and ‘specification process quality’ will determine the ‘adequacy of documentation for new functionality’ that is being implemented in this phase. If, for example, there is very little new functionality (and so the ‘scale of new specification and documentation work’ is low) then, even if the ‘specification process quality’ is poor, it is

likely that 'adequacy of documentation' will be sufficient. On the other hand, if there is a lot of new functionality the 'scale of new specification and documentation work' is likely to be high, which means that the 'specification process quality' will need to be good in order for the 'adequacy of documentation' to be sufficient.

4. The amount of 'new functionality implemented' will influence the inherent number of defects in the new code. We distinguish between potential defects from poor specification and hence we have the two nodes 'inherent potential defects from poor specification' and 'inherent potential defects (independent of specification)'. A very good quality specification can eliminate most of the former (the nodes 'probability of avoiding specification defect' and 'potential defects given specification and documentation adequacy' model this). However, the number of 'inherent potential defects (independent of specification)' is a function of the number of function points delivered (it is based on empirical data by Jones [1999]).
5. The number of 'total potential defects' that actually make it into the code is determined by the 'development process quality' (which characterises the overall quality of the development and coding process in this phase). Specifically, we use the 'probability of avoiding defect in development' to determine 'new defects in' given total potential defects. This number represents the number of defects (before testing) that are in the new code that has been implemented.
6. The next part of the BN deals with testing. If the new code we have developed is the first bit of code developed for this module then the number of previously existing defects in the module is zero. But generally the new code will be part of some existing code that contains some residual defects, hence the node 'residual defects PRE'. The 'total defects in' is therefore simply the sum of 'residual defects PRE' and 'defects in'. Suppose we do some testing. The number of 'defects found' is clearly dependent on the 'total defects in' and the 'probability of finding defect' (which is an output of the 'testing and rework process quality' subnet for this phase). If the testing process is very good then we are likely to find most of the defects. We may or may not decide to fix the defects found in testing in this phase; the success of such fixes will depend on the 'probability of fixing defect'. The 'total defects in' minus the 'fixed defects' leaves the number of 'residual defects POST'. We now use the posteriors of this node to replace the priors of the node 'residual defects PRE' for any subsequent phase in which further development and/or testing of **this module** takes place.

6 Evaluation of the Phase-Level Network

As we have mentioned, the phase level networks were developed using a combination of real-world data and experience. However, these models must clearly be validated against project data in order to build confidence in them. In this section, we will report on an extensive validation activity that has been performed at the Philips Software Centre, Bangalore. We should emphasise at the outset that this validation was performed using a single "all-activities" phase-level network, and this left the modelling open to one or two potential sources of inaccuracy. We will come back to this point towards the and of this section.

The questionnaire view of the interface to the all-activities model requires both quantitative and qualitative data. The quantitative data could all be provided from the standard metrics collection activities. Key people who had been involved in the projects provided the qualitative, more judgemental, data. This is an area where we are steadily building up experience and developing guidelines to enable accurate data is collected, with minimal inter-subject variation.

There were in all around 116 closed projects in CE. To narrow this down to a good sample, the Quality Managers were requested to provide the list of the projects in their respective LoBs that could be taken up for evaluation. A short list of criteria for the exclusion of projects was provided:

1. Data was unreliable
2. Project was not completed
3. No key persons available for the project

After excluding projects according to the above criteria, 41 projects remained that could be used for evaluating the all-activities model. A questionnaire was prepared and circulated to the identified key persons, usually SQEs, of these projects. The qualitative data for the respective projects was then compiled from the project database. Sufficient data was finally obtained on 31 projects to enable the all-activities

network to be used to make predictions of the number of defects that would be found during testing in each of these projects. These predictions were than compared with the actual numbers of defects found.

The actual comparison of predicted defects versus actual defects was performed in a two-round process. We report the results from both steps. The reason for needing to repeat the comparisons a second time was that in the first phase it was found that:

1. Some of the projects had not recorded all the defects found during testing. This was because later parts of the testing phase had been performed outside of PSC;
2. We originally took the qualitative inputs from one individual (per project). This could lead to a bias on the inputs due to the individual's perception. This was verified when we reviewed the inputs with the Quality Leaders during the round 2 exercise.

Prior to round two, more detailed guidelines were developed for answering the questionnaire. This lead to more "honest" answers to some of the questions where there had been a clear tendency to avoid any apparent criticism of some aspect of the execution of a project. Figure 6.1 summarises the results obtained from both rounds of the evaluation.

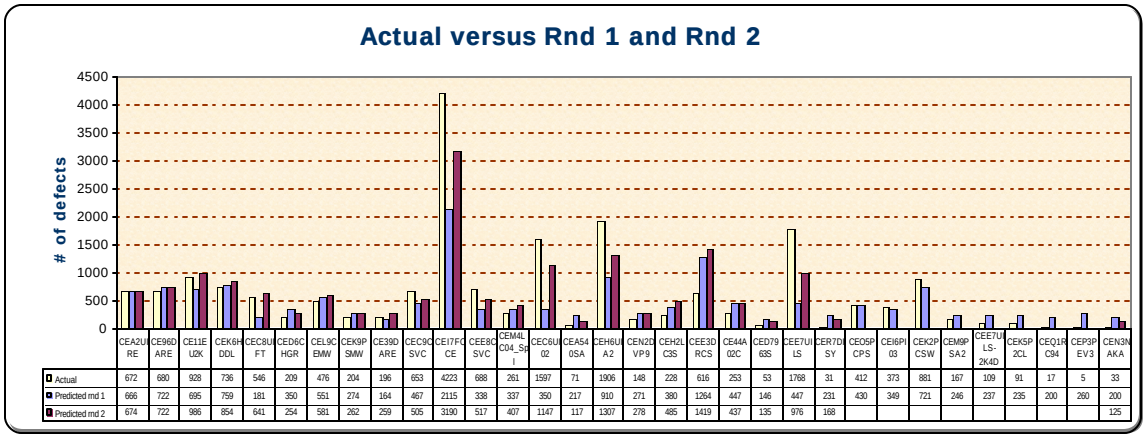


Figure 6.1: Actual versus predicted defects for the two evaluation rounds

It can be seen from this figure that the covariance between the actual and predicted defects for round two was very strong (note that at the time of writing some data was still awaited for round 2). This emphasises the importance of careful procedures for eliciting the qualitative data for the questionnaire. A range of techniques for reducing bias in such data is available [Meyer and Booker, 1991], but this is still an area of active research for us.

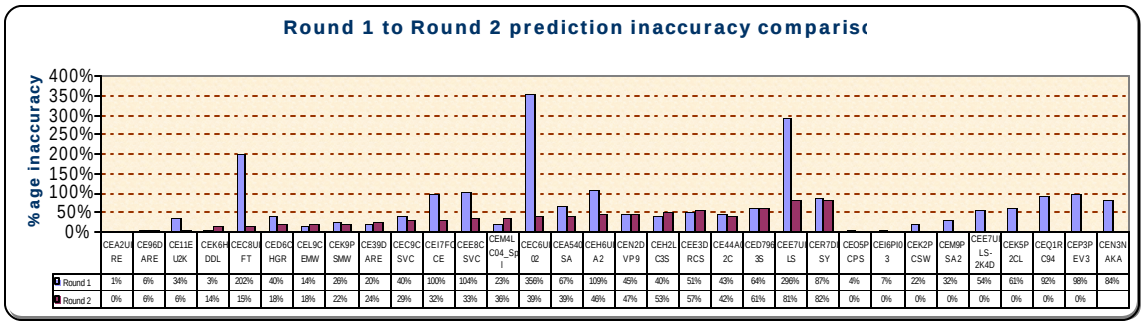


Figure 6.2: Prediction inaccuracy for the two evaluation rounds

Figure 6.2 summarises the percentage inaccuracy for the two rounds. As well as an improvement in the correlation between predicted and actuals, round two also showed an overall improvement in prediction accuracy. However, it can be seen that there are still a number of cases where the prediction inaccuracy is still very high. We investigated these results in a little more detail to identify some possible causes of these inaccuracies.

We looked at a number of aspects of these projects to see if there was some possible explanation for the variation in accuracy. The most promising avenue to explore came from the following summary of inaccuracy for different sizes of projects:

- For code size less then 5 KLOC the prediction inaccuracy was more than 70%
- For code sizes between 5 and 10 KLOC the prediction inaccuracy was between 40% and 70%
- For code sizes between 10 and 90 KLOC, the inaccuracies are less than or equal to 30% (with 2 outliers from these 17 projects
- For code sizes above 90 KLOC, the inaccuracies are between 40% and 80%

Overall the best predictions (< 20% inaccuracy) were observed for projects with between 50 and 87 KLOC. This clear influence of code size lead to a number of observations.

Firstly, the assessment models use function points as the primary measure of project size. KLOC and programming language are modelled as "indicators" of the total number of function points implemented. However, unless the value for function points is actually fixed, other project factors may influence the internal value for function points. Recommended practice is to initially enter the value of KLOC and choice of programming language, and then use the model to estimate the number of function points the KLOC figure corresponds to. The median for the estimated number of function points should then be entered as an observation.

Secondly, the all-activities model has an "input node" for the number of defects that may be present in the work product before the current phase of development. If the all-activities model is being used to model a complete project, then an observation of "0" should be explicitly entered for "defects pre". Otherwise, the model will use a prior-distribution for number of defects pre with a median of about 100. This will provide a very significant bias in the predictions of defects for smaller projects.

This problem was easily overcome within the modelling method we have described by explicitly modelling the pre-existing code, using a simple stub phase (no specification, development or testing), as shown in Figure 6.1:

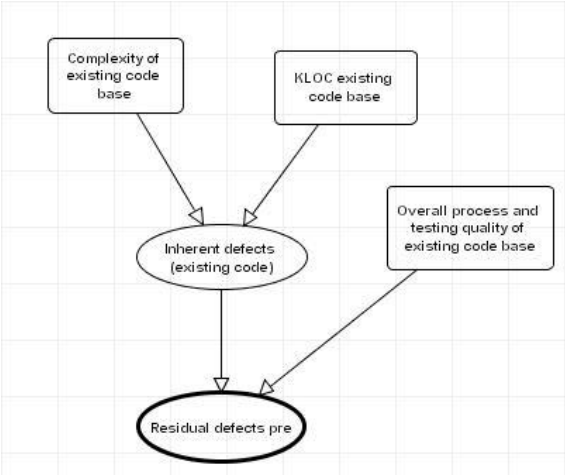


Figure 6.1 Stub phase for existing code

With this simple modification, we reran the evaluation with the following results:

- For code sizes between 10 and 90 KLOC, the predictions for defects found were exceptionally accurate (inaccuracies are less than 30%).
- The best predictions (inaccuracy <20%) were obtained for code sizes between 50 and 87 KLOC.
- For code size < 5 KLOC the prediction inaccuracy was more than 70%.
- For code sizes between 5 and 10 KLOC and greater than 90 KLOC, the prediction inaccuracy was between 40% and 80%.

The relative inaccuracies outside the range 10 to 90 KLOC were inevitable given that the default used has been configured only for code between 20 to 80 KLOC.

Perhaps the most impressive single statistic was figure for the correlation between predicted and actual values of 95%.

7 Conclusions

We have described a new approach to software quality control and defect prediction using Bayesian Networks. Our evaluation showed good results for projects with code sizes between 10 and 90 KLOC. Investigations indicate that we can improve the prediction accuracy at the lower and higher ends of the KLOC scale.

We have deliberately focused on evaluating the models for predicting the number of defects that will be found during testing of a product prior to release, as we have good data available for this. Although this is a valuable result in itself, the technique will obviously have even greater impact if we can use it to predict quality post-release. Evaluation of this aspect of the tool is ongoing work. However, preliminary results on this, and the work reported in this paper, give us confidence that we can achieve significant results in this area as well.

8 Acknowledgements

This report is based in part on work undertaken on the following funded research projects: MODIST (EC Framework 5 Project IST-2000-28749), SCULLY (EPSRC Project GR/N00258), SIMP (EPSRC Systems Integration Initiative Programme Project GR/N39234), and SCORE (EPSRC Project Critical Systems Programme GR/R24197/01). We also acknowledge the contributions of individuals from Israel Aircraft Industries, QinetiQ and BAE Systems.

We would like to thank the development team at Agena Ltd for their work on AgenaRisk and the MODIST prototypes that were used in the early stages of this work. We would also like to thank Hans Aerts and Wilko van Asseldonk for mentoring the project and supporting the final evaluations. Finally we would like to thank Geert Acke for providing valuable additional evaluations that help to refine the final result.

9 References

Agena Ltd, "*Bayesian Belief Nets*", http://www.agena.co.uk/bbn_article/bbns.html, 1999.

N.E. Fenton and S.L. Pfleeger, *Software Metrics: A Rigorous and Practical Approach*, (2nd Edition), PWS Publishing Company, 1997.

N. Fenton and M. Neil "A Critique of Software Defect Prediction Research", *IEEE Trans. Software Eng.*, **25**, No.5, 1999.

N. Fenton and N. Ohlsson "Quantitative analysis of faults and failures in a complex software system", *IEEE Trans. Software Eng.*, **26**, 797-814, 2000.

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

HUGIN Expert Brochure. Hugin Expert A/S, P.O. Box 8201 DK-9220 Aalborg, Denmark, 1998.

IMPRESS (IMproving the software PRocESS using bayesian nets) EPSRC Project GR/L06683, http://www.csr.city.ac.uk/csr_city/projects/impress.html, 1999.

Jones C, "Software sizing", *IEE Review* 45(4), 165-167, 1999.

P.J. Krause. "Learning Probabilistic Networks", *Knowledge Engineering Review*, **13**, 321-351, 1998

S.L. Lauritzen and D.J. Spiegelhalter, "Local computations with probabilities on graphical structures and their application to expert systems (with discussion)" *J. Roy. Stat. Soc.* **Ser B 50**, pp. 157-224, 1988.

N. Lewis, "Continuous process improvement using Bayesian Belief Networks. The lessons to be learnt". *Proceedings of the twenty forth international conference on Computers and Industrial Engineering*. Brunel University. 9th-11th September, 1998.

[28] M. A. Meyer and J. M. Booker, *Elicitating and Analyzing Expert Judgement: A Practical Guide*, Academic Press, Ltd., 1991.

McCall, P.K. Richards and G.F. Walters, *Factors in software quality. Volumes 1, 2 and 3*. Springfield Va., NTIS, AD/A-049-014/015/055, 1977.

J. Musa, *Software Reliability Engineering*, McGraw Hill, 1999.

M. Neil, B. Littlewood and N. Fenton, "Applying Bayesian Belief Networks to Systems Dependability Assessment". *Proceedings of Safety Critical Systems Club Symposium*, Leeds, Published by Springer-Verlag, 6-8 February 1996.

M. Neil, N. Fenton and L. Nielson, "Building large-scale Bayesian Networks", *Knowledge Engineering Review, to appear* 2000.

J. Pearl, *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference* Morgan Kauffman, 1988. (Revised in 1997)

SERENE consortium, "SERENE (SafEty and Risk Evaluation using bayesian Nets): Method Manual", ESPRIT Project 22187, http://www.dcs.qmw.ac.uk/~norman /serene.htm, 1999.