IOSR Journal of Computer Engineering (IOSR-JCE)
e-ISSN: 2278-0661, p- ISSN: 2278-8727Volume 8, Issue 5 (Jan. - Feb. 2013), PP 52-62
www.iosrjournals.org

Software Quality Modelling Using Bayesian Networks

Swati Agrawal[1], P.C.Gupta[2]
Computer & Engg. Deptt. Jaipur National University, Jaipur-302018 India

Abstract: *This research work provides an introduction to the use of Bayesian Network(BN) models in Software Engineering. A brief overview of the of BNs is included, together with an explanation of why BNs are ideally suited to dealing with the characteristics and shortcomings of typical software development environments. This theory is illustrated using real world models that illustrate the advantages of BNs in dealing with uncertainty, causal reasoning and learning in the presence of limited data.*
Keywords *– BN, COCOMO, COQUALMO, CI, CPD, modist.*

I. INTRODUCTION

Software project planning is notoriously unreliable. Attempts to predict the effort, cost and quality of software projects have foundered for many reasons. These include the amount of effort involved in collecting metrics, the lack of crucial data, the subjective nature of some of the variables involved and the complex interaction of the many variables which can affect a software project.**[1]**
Research cover sufficient BN theory to enable the reader to construct and use BN models using a suitable tool. The statistical nature of BN models automatically enables them to deal with the uncertainty and risk that is inherent in all but the most trivial software projects.
Two idiosyncratic types of model will be presented. The first group is causal in nature. These take results from empirical software engineering, and using expert domain knowledge, construct a network of causal influences. Known evidence from a particular project is entered into these models in order to predict desired outcomes such as cost, effort or quality.

The second group of models is primarily parameter learning models for use in iterative or agile environments. By parameter learning we mean that the model learns the uncertain values of the parameters as a project progresses and uses these to predict what might happen next. They take advantage of knowledge gained in one or more iterations of the software development process to inform prediction so later iterations.

Background

Before we can describe BN software project models, it is worthwhile examining the problems that such models are trying to address and why it is that traditional approaches have proved so difficult. Then, by introducing the basics of BN theory.

II. COST AND QUALITY MODELS

Software process models can be categories as cost models and quality models. Cost models, aim to predict the cost of a software project. Similarly, since the "size" of a software project often has a direct bearing on the effort and cost involved, we also include project size models in this category. Quality models are concerned with predicting quality attributes such as mean time between failures, or defect counts.

Estimating the cost of software projects is notoriously hard. Molokken and Jorgensen (2003) performed a review of surveys of software effort estimation and predict that the average cost overrun was of the order 30-40%. One of the most famous such surveys, the Standish Report(Standish Group International 1995) puts the mean cost overrun even higher, at 89%. Software quality prediction, has been no more successful. Fenton and Neil (1999) have described the reasons for this failure. Research briefly reproduces these here since they apply equally to both cost and quality models.
1. The cost and quality models, such as COCOMO (Boehm 2010) and COQUALMO (Chulani & Boehm 1999) take one or two parameters which are fed into a simple algebraic formula and predict a fixed value for some desired cost or quality metric. They are therefore unable to attach any measure of risk to their predictions. Changes in parameters and coefficients can be simulated in an ad-hoc fashion to try to address this, but this is not widely used and does not arise as a natural component of the base model.[9,11]
2. Parametric models cannot easily deal with missing or uncertain data. This is a major problem when constructing software process models. Data can be missing because it simply wasn't recorded. It can also be missing because the project is in a very early stage of its lifecycle. Some of the problems appear quite prosaic, for example ambiguity arises because of difficulties in defining what constitutes a line of code.
3. Traditional models have difficulty incorporating subjective judgments, yet software development is replete

with such judgments. The cost and quality of a software project clearly depend to a significant extent on the quality of the development team.
4. Parametric models typically depend on a previous metrics measurement programme. A consistent and comprehensive metrics programme requires a level of discipline and management commitment which can often evaporate as deadlines approach and corners are cut. Failure to adjust the coefficients in a parametric model to match local conditions can result in predictions which are significantly (Briand et. al. 1999; Kemerer 1987) [10].
5. Metrics programmes may uncover a simple relationship between an input and an output metric, but they tell us nothing about how this relationship arises, and crucially, they do not tell us what we must do to improve performance. Many attempts have been made to find alternatives to simple parametric models. These include multivariate models (Neil1992), classification and regression trees (Srinivasan & Fisher 1995), analogy based models (Shepperd & Schofield 1997; Briand et. al. 1999), artificial neural networks (Finnie & Wittig & Desharnais 1997) and systems dynamics models(Abdel-Hamid 2008)[2,12,15].

III. Introduction To Bayesian Networks

A Bayesian Network (BN), (Jensen, 2001) [18], is a directed acyclic graph shown in Figure 1. Nodes without parents such as the “Probability of finding defects” and “Defects In” nodes in Figure 1 are defined through their prior probability distributions. Nodes with parents are defined through Conditional Probability Distributions (CPDs). For some nodes, the CPDs are defined through deterministic functions of their parents (such as the “Defects Out” node in Figure 1. Conditional independence (CI) relationships are implicit in the directed acyclic graph: all nodes are conditionally independent of their ancestors given their parents, within comparison with regression based models there are a number of beneficial features and advantages:
1. One of the most obvious characteristics of BNs is that we are no longer dealing with simple point value models. In particular, its predictions, in this case limited to the “Defects out” node, is in the form of a marginal distribution which is typically unimodal, giving rise to a natural “most likely” median value and a quantitative measure of risk assessment in the form of the posterior marginal’s variance.
2. The model then uses the prior distributions which can be based on empirical studies of a wide range of software projects, such as those provided by Jones (1986; 1999; 2003), or by publicly available databases (ISBSG 2008) [16,20].
3. The nodes in this model are all represented by numeric scales. BNs are not limited to this however. Any set of states which can be assigned a probability can be handled by a BN.
4. Unlike parametric models, where the underlying variation in software process measurement has been “averaged” out, this model contains all of the available information. It is therefore not limited to an “average” project, but can encompass the full range of values included in its priors.

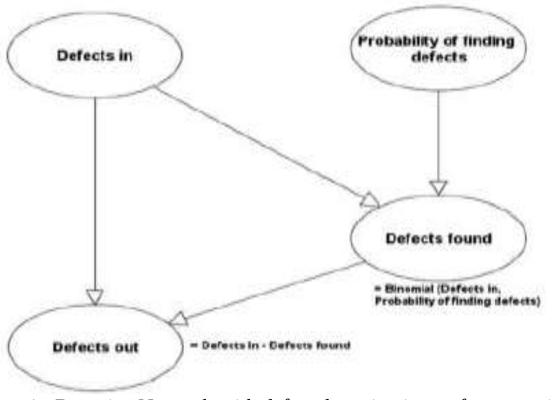

Defects in
Probability of finding defects
Defects found
= Binomial (Defects in, Probability of finding defects)
Defects out
= Defects in - Defects found

Figure 1 : Bayesian Network with defect detection in a software project

5. Unlike simple regression models the relationship between all the causal factors is explicitly stated. This means that, in addition to using the model to make predictions, we can set our desired outcome (for example to how many defects are acceptable in the released product) and the model will be able to tell us how many “Defects in” are the most likely explanation of this and what our probability of finding a defect must be.
The presence of an observation updates the CPD of its children and, through Bayes theorem, the distributions of its parents. In this way observations are propagated recursively through the model. BN models can therefore update their beliefs about probable causes and so learn from the evidence entered into the model.(Jensen (2001) and Lauritzen & Spiegelhalter (1988).

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)} \qquad (1)$$

The models presented here are all first order Markov. Informally this means that the future is independent of the past given the present $P\left(Z_t \mid Z_{1,t-1}\right) = P\left(Z_t \mid Z_{t-1}\right)$and in practice it means that we do not need to re-compute the model afresh each time a new prediction is needed. The first order Markov property reduces the number of dependencies, making it computationally feasible to construct models with a larger numbers of timeslices. Consistent propagation is achieved using standard junction tree algorithms(Lauritzen & Spiegelhalter 1988). These algorithms provide exact (as opposed to approximate) propagation in discrete BNs and are generally regarded as among the most efficient BN propagation algorithms (Lepar & Shenoy 1998).

INDICATOR NODES AND RANKED NODES

Two types of nodes deserve special mention these are indicator nodes and ranked nodes. Indicator nodes are nodes with no children and a single parent. They are often used in circumstances where the parent is not directly observable but where some indicator of the parent's status can be measured easily (hence their name).

Indicator nodes can also be used where a large number of causal factors all have a direct impact on a single child node. The number of entries in the CPD of the child grows exponentially with the number of parents.

Ranked nodes are nodes whose states are measured on an ordinal scale, often with either three or five states ranging from "Very Low" through to "Very High". They are used to elicit subjective opinions from experts so that they can be entered as observations into the model. In the tool used to build most of the models described here (Agena Ltd. 2008)[4].

THE MODIST MODEL

Fenton and Neil's pioneering paper (Fenton & Neil1999)[13] inspired a number of research groups to apply BNs to software process modelling. Wooff, Goldstein, and Coolen (2002) have developed BNs modeling the software test process while Stamelos etal(2003)usedCOCOMO81costfactorstobuild a BN model of software project productivity. Bibi and Stamelos(2004) have shown how BNs can be constructed to model IBM's Rational Unified Process[8,14]. Fenton and Neil's own research group have also gone on to develop a series of BN models, culminating in the AID tool (Neil, Krause, & Fenton 2003), the MODIST models (Fenton et. al. 2004), and the trials of revised MODIST models at Philips (Neil & Fenton 2005; Fenton et. al 2007a; Fenton et. al 2007b). A similar model has been developed by Siemens (Wang et. al. 2006).

A greatly simplified version of the MODIST model, with many of the causal factors and indicator nodes removed, is shown in Figure 2. The subnets are:

• Distributed communications and management. Contains variables that capture the nature and scale of the distributed aspects of the project and the extent to which these are well managed.

• Requirements and specification. Contains variables relating to the extent to which the project is likely to produce accurate and clear requirements and specifications.

• Process quality. Contains variables relating to the quality of the development processes used in the project.

• People quality. Contains variables relating to the quality of people working on the project.

• Functionality delivered. Contains all relevant variables relating to the amount of new functionality delivered on the project, including the effort assigned to the project.

• Quality delivered. Contains all relevant variables relating to both the final quality of the system delivered and the extent to which it provides user satisfaction (note the clear distinction between the two).

PEOPLE QUALITY SUBNET

Figure 3 shows the variables and causal connections in this subnet. A description of the main variables, including the model's rationale is given in Table 1. All of the variables shown in Figure 3 are ranked nodes, measured using a five point scale ranging from very low to very high. Observations are not normally entered directly on the variables described in Table 1. Instead we enter observation at primary causes (variables with no parents in the BN) and indicators.

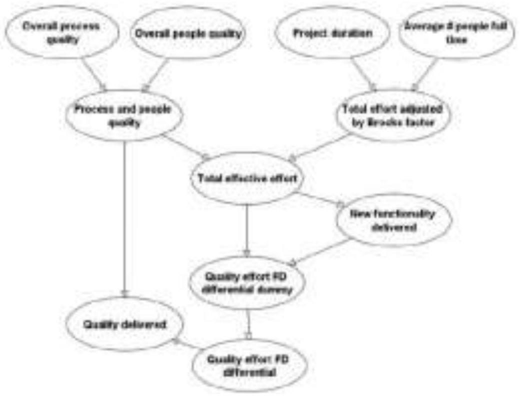

Overall process quality
Overall people quality
Project duration
Average # people full time
Process and people quality
Total effort adjusted by Brooks factor
Total effective effort
New functionality delivered
Quality effort FD differential dummy
Quality delivered
Quality effort FD differential

Figure 2 : A simplified version of the MODIST model

USING THE BAYESIAN NET FOR DECISION SUPPORT

What makes the Bayesian resource model so powerful, when compared with traditional software cost models, is that we can enter observations anywhere to perform not just predictions but also many types of trade-off analysis and risk assessment.

As an example of this, if we simply set "New functionality delivered" to 1000 function points, the model produces the predictions shown in Figure 4. This tells us that the most likely duration is 6 to 8 months (the modal value), although the mean and median values are considerably larger at 21 months and 17 months respectively

If we fix the number of people at 10 and set the "Process and people quality" to "Very High", we get a modal value for the duration of 2 to 4 months, and mean and median values of 10 and 5 months respectively with a standard deviation of 13 months, making the model far less uncertain in its predictions. These figures are consistent with typical parametric models. However, this ability to vary the project constraints in any desired way while producing consistent estimates of the risk, provides far greater insight into the interplay between variables and allows far greater flexibility than is possible with parametric models.

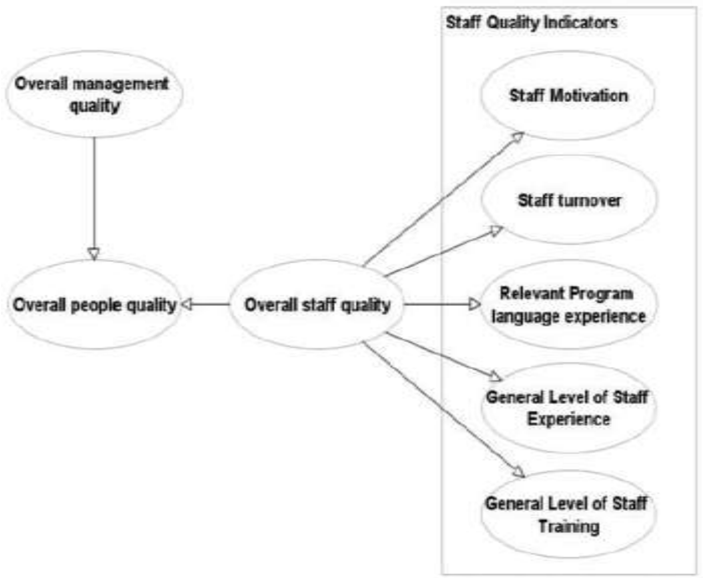

Staff Quality Indicators
Overall management quality
Overall people quality
Overall staff quality
Staff Motivation
Staff turnover
Relevant Program language experience
General Level of Staff Experience
General Level of Staff Training

Figure 3: The "People quality" subnet of the MODIST model

THE DEFECT MODEL

As with the MODIST model, the defect model (Neil & Fenton 2005) is too large to be shown here in full. The core of the model is shown in Figure 5.

Separate subnets governing: the scale of new functionality, specification and documentation quality, design and development quality, testing and rework quality, and features of the existing code base, all feed into this core defect model. Unlike the MODIST model, the defects model does not require all of its subnets to be included. The defects model is designed to model multiple phases of software development where not all aspects of the full software development process are present in each development phase.

Table 1: Details of subnet for people quality

Variable Names	Description
Overall management quality	This is a synthetic node that combines 'Communications management adequacy', sub-contractment adequacy' and 'interaction management adequacy'. If any of these three is poor then general value of 'overall management quality' will be poor.
Overall staff quality	This is the quality of non-management staff working on the project.
Overall people quality	This is a synthetic node that combines 'overall management quality', and 'overall staff quality'.

The model operates as follows. A piece of software of a given size, in this case measured inn thousands of lines of code (KLOC) gives rise to a certain number of defects. Some of these defects will be the result of inaccuracies in the specification and requirements.

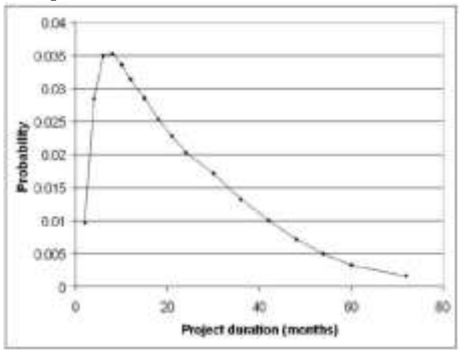


Figure 4 : Probability distribution of "Project duration" for 1000 FPs

The model shown in Figure 5 actually comes from a model validated at Philips (Neil & Fenton 2005; Fenton et. al 2007a; Fenton et. al 2007b). This differed from the original defects model in several important respects.

Project size was measured in KLOC. The original defects model used function points (FPs) as its size measure. However, FPs were not widely used at Philips and it seemed unlikely that they could be introduced purely to validate this model.

Initialling the model was simply modified to deduce the number of FPs from the KLOC. This led to problems because it introduced a level of uncertainty in the FP measure which was not originally present. FPs also include a measure of problem complexity which is absent from the KLOC measure.

The second big change from the defects model was that many of the original indicator nodes were converted to causal factors.

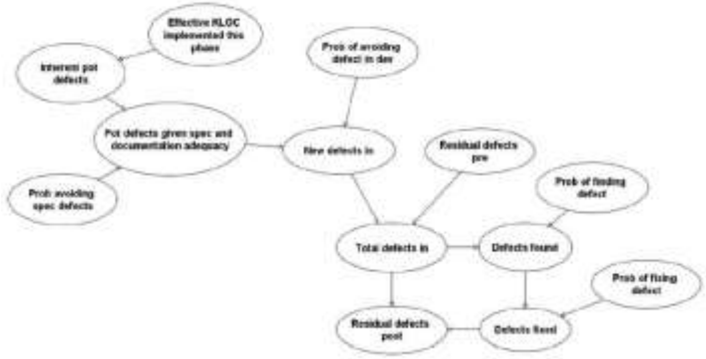


Figure 5 : The core of the defects model

The model has been extensively trialed at Philips, with considerable success. The R_2 correlation between predicted and actual defect values is in excess of 93%. It is of course possible to construct regression models with similar, or even higher correlations, but only by including most of the data as training data, which implies an extensive metrics collection programme. (Fenton et. al 2007b).

IV. The Productivity Model

While the MODIST and defects models were successful in practical applications they contain some limitations. Overcoming or reducing these limitations became the motivation for the Productivity Model. This model provides unique features which were not available in previous models:

1. This model permits custom prior productivity rates and custom prior defect rates. Companies typically gather some data from their past projects. Productivity and defect rates are among the easiest to extract from project databases.
2. This model enables trade-off analysis between key project variables on a numeric scale. All the key project variables, namely: functionality, software quality and effort are expressed as numbers.
3. This model enables customised units of measurement. Previous models captured key variables expressed in fixed units of measurement.
4. The impact of qualitative factors can be easily changed by changing weights in node expressions. We provide a questionnaire which can help determine users' opinions on the relationships between various model variables.
5. This model allows target values for numeric variables to be defined as intervals, not just as point values. For example, this model can answer the question: how can we achieve a defect rate between 0.05 and 0.08 defects/ FP for a project of a specific size and with other possible constraints.
6. Numeric variables in this model are dynamically discretised. This means that setting intervals for numeric variables occurs automatically during model calculation.

There are three steps which need to be followed before the Productivity Model can be used in performing analyses:

Step 1: Calibrate the model to the individual company. This step involves adding new detailed factors or removing those which are not relevant.

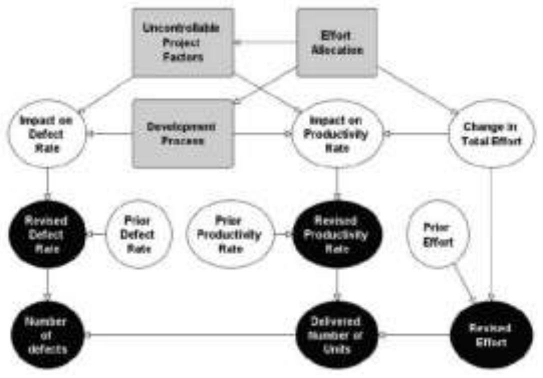


Figure 6 : The core of the Productivity Model

Step 2: Nominate a typical past project from the past project database. The user nominates a past project developed by this company which is closest to the project to be modelled.
Step 3: Estimate the difference in the current project. This step involves assessing how the current project is different from the project(s) nominated in step 2.

USING PRODUCTIVITY MODEL IN DECISION SUPPORT

Let us suppose that a software company has to deliver software consisting of 500 function points but constrained to 2500 person-hours of effort. Suppose that in similar projects in the past the defect rate was typically 0.15 defects per function point, productivity rate was 0.2 function points per person-hour and the effort was 2000 person-hours.

Now let us further assume that the company is not able to improve the process and people for this project. Thus we need to perform trade-off analysis between key project variables. We might remove a constraint for the product size. This would result in predicting lower product size containing lower total number of defects. But sacrificing product size would rarely be a good solution. We rather analyze how much more effort is required to achieve the target for the number of defects. The model now predicts that this company should spend around 7215 person-hours on this project to achieve the lower number of residual defects after

release. The majority of the increased effort should be allocated to specification and testing activities.
In addition to the higher project complexity, suppose we also assume that there is the highest possible deadline pressure. In this case, increased deadline pressure causes the developers to work faster and thus become more productive (0.176 FP/ person-hour). However, it also means they are less focused on delivering software of similar quality and thus their revised defect rate is expected to further increase (0.073 defects/FP).

Let us now assume that in addition to the previously entered known project factors they anticipate receiving input documentation of higher quality. The model predicts that with increased quality of input documentation we should expect to be more productive (0.180 FP/ person-month) and deliver better software (0.067 defects/FP).

MODELLING PRIOR PRODUCTIVITY AND DEFECT RATES

The PDR model aims to estimate prior productivity and defect rates which are then passed as inputs to the Productivity Model. This model has a Naïve Bayesian Classifier structure (Figure 7). Observations are entered into qualitative factors describing the software to be developed (white ellipses). The model then estimates the log of productivity and defect rates. Finally, dependent variables (productivity and defect rates) are calculated on their original scale.

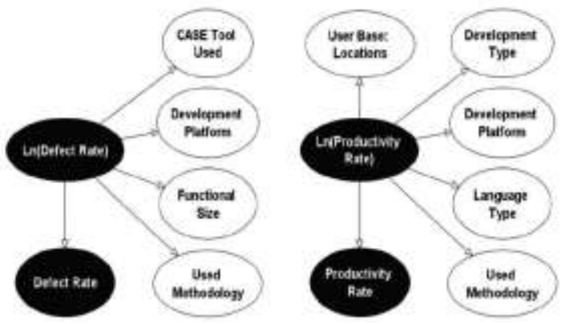


Figure 7 : PDR Model

After running both models we observe that the observations entered into the PDR model significantly change the predictions provided by the Productivity Model. Predicted revised effort is about 3.5 times higher in scenario 1 (10740 person-hours) than in scenario 2 (3074 person-hours). However, it would be wrong to conclude from this that we should only use application generators and multiple platforms, while avoiding 3GLlanguages and mainframe platforms.

V. Agile Development Environments

The models discussed so far mostly apply to large, traditional, waterfall style development environments. Agile methods eschew extensive specification and design phases in favour of rapid prototyping, iterative development, extensive customer feedback and a willingness to modify requirements in the face of changing business circumstances. Agile development environments present two problems for traditional models. The first problem is that the lack of a formal requirements gathering phase makes it very difficult to quantify the size of the project [5]. The second problem that traditional models face concerns data gathering and entry. The defects model described in an earlier section can require over 30 individual pieces of data in order to be fully specified. Although not all data is required in all phases of development, there will normally be a need to gather several separate numbers or subjective values.

This combination of intelligent model adaptation, and learning using minimal data input, is only possible because of the empirically derived priors and the causal relationships elicited from domain experts. The BN models already represents a very wide range of possible development environments. Rather than "training" the model, data is needed to simply "nudge" the model towards a set of variable states that are consistent with the environment being modelled.

VI. Extreme Programming Project Velocity Model

Extreme Programming (XP) is an agile development method that consists of a collection of values, principles and practices as outlined by Kent Beck, Ward Cunningham and Ron Jeffries (Beck 2010;[7] Jeffries, Anderson & Hendrickson 2000) [17]. These include most notably: iterative development, pair programming, collective code ownership, frequent integration, onsite customer input, unit testing, and refactoring.

This is the Project Velocity V_i for iteration i. Assuming that the next iteration, i + 1, is the same length, the customer selects the highest priority uncompleted user stories whose estimated efforts sum to $V_{i.}$ These user stories are then scheduled for iteration i + 1. The project velocity can therefore be thought of as the estimated productive effort per iteration.

The BN used to model project velocity is shown in Figure 8. To model the relationship between total effort E_i and the actual productive effort A_i, there is a single controlling factor that we call Process Effectiveness, e_i. This is a real number in the range[0,1]. A Process Effectiveness of one means that all available effort becomes part of the actual productive effort. The actual productive effort is converted into the estimated productive effort (or project velocity) V_i, via a bias b_i, which represents any consistent bias in the team's estimations.

Only E_i and V_i are ever entered into the model. The model shown in Figure 8 is what is known as a 1.5 TBN (for 1.5 time slice temporal BN). It shows the interface nodes from the previous time slice and their directed arcs into the current time slice. This is essentially the full model, it is not a cut down core such as the ones presented for the MODIST and defect models. Notice the tiny size of this mode compared to the others. This is due to the fact that it is only trying to predict one thing, Vi, and that is does so, not by taking into account all possible causal factors, but by learning their cumulative effect on the process control parameters: l_i and r_i.

The model works as follows. Initial estimates for the amount of available effort E_i are entered into the model for each iteration. At this stage the number of iterations is unknown, so a suitably large number must be chosen to be sure of covering the entire project. Using empirically derived industry priors for l_0, r_0, e_0 and b_0 the model then makes generic predictions for the behaviour of V_i. This enables the project management to see how many user stories are likely to be delivered in each iteration. The model correctly reproduces rapidly rising values for V_i over the initial iterations - a phenomenon that has been observed in multiple studies (Ahmed, Fraz, & Zahid 2003; Abrahamsson & Koskela, 2004; Williams, Shukla, & Anton, 2004)[3,6].

At each project iteration measured values for V_i become available and are entered into the model. This causes the learned parameters l_i, r_i and b_i to update, modifying future predictions. Using data from a real XPproject (Williams, Shukla, & Anton, 2004), we were able to enter observations for V_1 and V_2, and verify the model's predictions of the remaining V_i. Initially this generates improved predictions for early iterations, but significantly worse predictions for later iterations. It turns out that there was a significant change in the environment half way through the project, when much better customer contact and feedback became available. This was added to the model by adding an "Onsite customer" indicator node to l_i.

The model is not limited to productivity predictions. By adding a node which sums the V_i distributions, we can create predictions of delivered functionality, s_i, after each iteration.. Taking the median values of the s_i distributions and comparing them to the actual functionality delivered, we can determine the magnitude of relative errors (MRE) for each s_i. The mean values

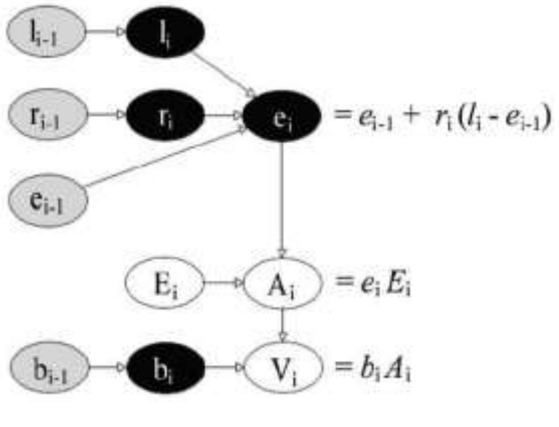


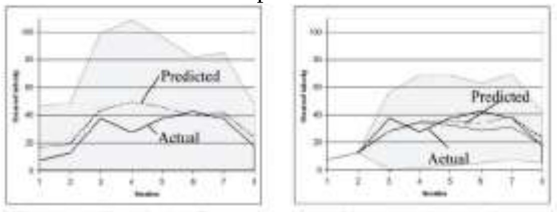
Figure 8 : Project Velocity model

of the MREs give a good overall measure of the accuracy of the model. The mean MRE for s_i before learning was 0.51, an error of over 50%. After learning the mean MRE for s_i reduces to 0.026 - an extraordinary level of accuracy for a software process model. One of the great advantages of a BN model is its ability to deliver natural assessments of risk. If we take the cumulative probability distribution of an s_i node, then this allows us to read off the probability that any given amount of functionality will be delivered.

XP is the most common agile development method. Another common methodology is Scrum (Takeuchi & Nonaka 1986; Schwaber & Beedle 2002; Sutherland 2004). This approach uses burn-down charts to plan a project. A burndown chart starts with a fixed amount of functionality that must be delivered. This reduces with each iteration as more and more functionality is completed. The slope of the burndown chart gives the project velocity, while its intercept with the horizontal time axis gives the projected completion date.

VII. Learning Model For Iterative Testing And Fixing Defects

The models discussed earlier contain fixed relationships between variables.



Predicted and actual median velocity values before any actual values have been entered into the model. The solid grey area shows the predicted values +/- 2 sd.

After v1 and v2 have been entered, and improved customer access has been added, the predictions improve significantly. The uncertainty surrounding the model's predictions also decrease. The solid grey line shows the predictions after v1 and v2, but before customer access has been added.

Figure 9 : Project velocity predictions before and after learning

that it learns the impact of particular predictors on dependent variables using a set of past data. The aim for this model is to predict the number of defects found and fixed in an iterative testing and fixing process. This iterative process assumes that all functionality has been developed prior to testing. The testing and fixing process is divided into series of iterations lasting a variable amount of time. More information on earlier versions of this model and its assumptions can be found in (Radlinski et al. 2008b).

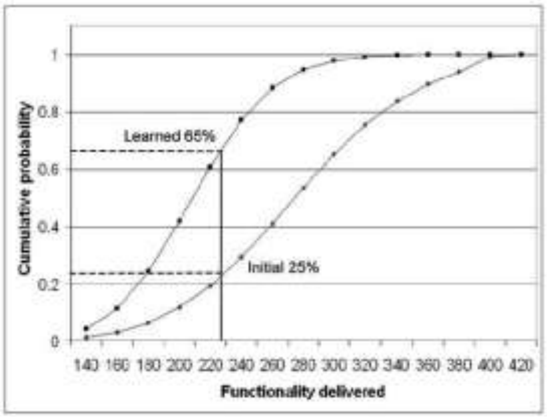


Figure 10 : Cumulative probability distribution of functionality delivered in iteration 8

The model works in the following way. A user enters data about the past testing and fixing iterations as well as the defects found and fixed in these iterations. The model uses the entered data to estimate its parameters – multipliers for each process factor and the number of residual defects.

We perform model validation using a semi-randomly generated dataset. We set point values for prior residual defects and values of process factor multipliers. Values for process factors were generated randomly and then manually adjusted in some iterations to more realistically reflect the testing and fixing process. The effectiveness

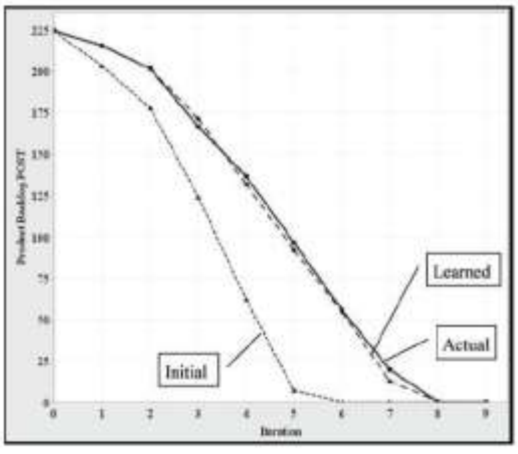


Figure 11 : Scrum burndown chart, before and after learning

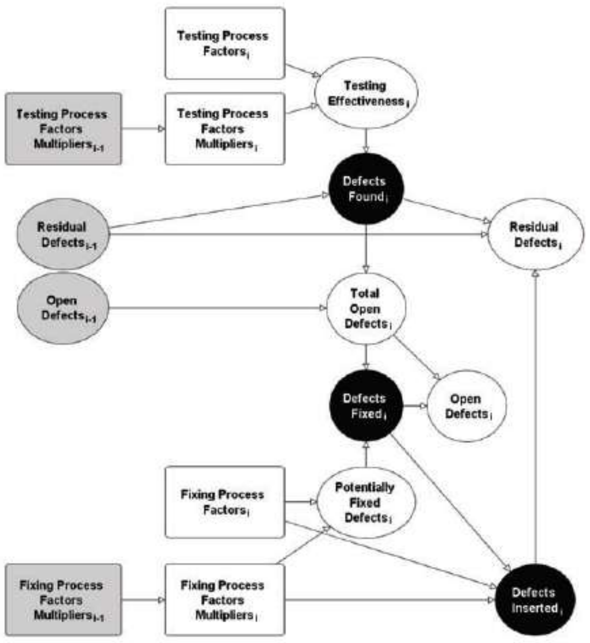

Testing Process Factors$_i$
Testing Process Factors Multipliers$_{i-1}$
Testing Process Factors Multipliers$_i$
Testing Effectiveness$_i$
Defects Found$_i$
Residual Defects$_{i-1}$
Residual Defects$_i$
Open Defects$_{i-1}$
Total Open Defects$_i$
Defects Fixed$_i$
Open Defects$_i$
Fixing Process Factors$_i$
Potentially Fixed Defects$_i$
Fixing Process Factors Multipliers$_{i-1}$
Fixing Process Factors Multipliers$_i$
Defects Inserted$_i$

Figure 12 : The structure of the LMITFD

We estimated values for defects found, fixed and inserted using the values of process factors and the relationships as incorporated by the model. We used these generated datasets in the model validation stage and treated them as if they were observed values. In the validation stage we tested how fast the model can learn the number of residual defects and the values of process factors' multipliers.
We tested the model using 30 testing and fixing iterations. First we used a single learning iteration with 29 iterations where values of defects found and fixed were predicted. This was followed by 2 iterations for learning and 28 for prediction, and so on. Figure 13 illustrates the values of relative errors in predicted total number of defects found and defects fixed as estimated after a different number of learning iterations. This relative error is defined as illustrated on Equation 2.

$$relative\,error = \frac{|total\ \ predicted - total\,actual|}{total\ \ actual} \qquad (2)$$

These results confirm that after only 5 learning iterations, the model predicts the total number of defects found to within a 0.16 relative error of the actual value and predicts total number of defects fixed to within a 0.30 relative error of the actual value. Predictions then become less accurate because of higher fluctuations in actual number of defects found in the dataset. But from 8 learning on number of iterations used to learn the model

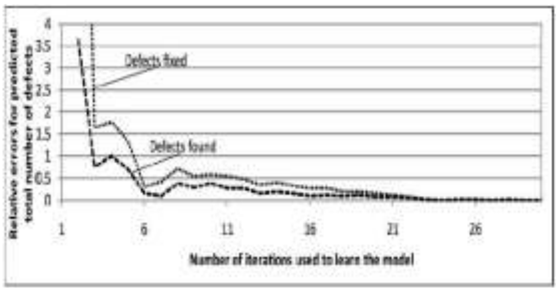
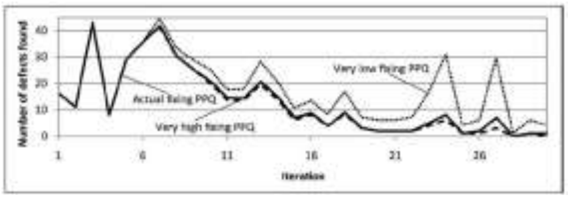


Figure 13 : Relative errors in predictions for total number of defects found and defects fixed depending iterations

One of the most important advantages of such models is the high potential for performing various types of what-if analyses. After the model learns its parameters, we can set as observations different values for future process factors to see how they influence predicted number of defects found. For example let us analyze model predictions in two scenarios when fixing process and people quality is 'very low' or 'very high'. Values of other process factors are the actual values in iterations used for prediction. We use 5 learning iterations and the remaining 25 for prediction. Figure 14 illustrates the model's predictions for this case. We can observe that, as we could expect, with lower fixing process and people quality, fewer defects are likely to be fixed in future iterations.

We can see that, although we have only modified our observations of fixing process and people quality, the predicted values of defects found are also different in these two scenarios.
Another issue which may be surprising in the beginning is the fact that predicted number of defects fixed in the late iterations is lower both with 'very low' and 'very high' fixing process and people quality compared with the scenario with the original fixing process and people quality. But there is also an explanation for such predictions. With 'very low' fixing process and people quality it is simply not possible to fix more defects without assigning significantly more fixing effort.



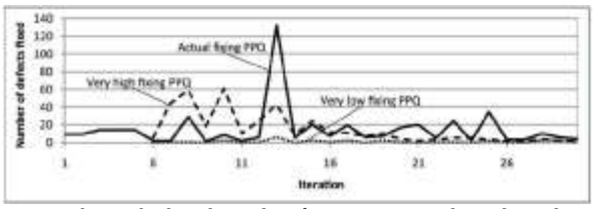


Figure 14 : Predictions from LMITFD with very high and very low fixing process and people quality after 5 iterations used to learn the model

References

[1]. Farid Meziane, University of Salford, UK, Sunil Vadera, University of Salford, UK, "Artificial Intelligence Applications for Improved Software Engineering Development: New Prospects", ISBN 978-1-60566-758-4, 2010.
[2]. Abdel-Hamid, T. (2008). The dynamics of software projects staffing: A system dynamics based simulation approach. IEEE Transactions on Software Engineering, 15(2), 109–119. doi:10.1109/32.21738
[3]. Abrahamsson, P., & Koskela, J. (2004). Extreme programming: A survey of empirical data from a controlled case study. In Proceedings 2004 International Symposium on Empirical Software Engineering, 2004. (pp. 73-82). Washington, DC: IEEE Computer Society.
[4]. Agena Ltd. (2008). Bayesian Network and simulation software for risk analysis and decision support. Retrieved July 9, 2008, from http://www. agena.co.uk/
[5]. Agile Manifesto. (2008). Manifesto for agile software development. Retrieved July 18, 2008, from http://www.agilemanifesto.org/
[6]. Ahmed, A., Fraz, M. M., & Zahid, F. A. (2003). Some results of experimentation with extreme programming paradigm. In 7thInternationalMulti Topic Conference, INMIC 2003, (pp. 387-390).
[7]. Beck, K. (2010). Extreme programming explained: Embrace change. Reading, MA: Addison-Wesley Professional.
[8]. Bibi, S., & Stamelos, I. (2004). Software process modeling with Bayesian belief networks. In 10th International Software Metrics Symposium Chicago.
[9]. Boehm, B. (2007). Software engineering economics. Englewood Cliffs, NJ: Prentice-Hall.
[10]. Briand, L. C., El Emam, K., Surmann, D., Wieczorek, I.,&Maxwell,K.D.(1999). An assessment and comparison of common software cost estimation modeling techniques. In 21st International Conference on Software Engineering, ICSE 1999, (pp. 313-322).
[11]. Chulani, S., & Boehm, B. (1999). Modeling software defect introduction and removal: COQUALMO (Constructive Quality Model),(Tech. Rep. USC-CSE-99-510). University of Southern California, Center for Software Engineering, Los Angeles, CA.
[12]. Fenton, N., Neil, M., Marsh, W., Hearty, P., Marquez, D., Krause, P., & Mishra, R. (2007a, January). Predicting software defects in varying development lifecycles using Bayesian nets. Information and Software Technology, 49(1),32–43. doi:10.1016/j.infsof.2006.09.001
[13]. Fenton, N., Neil, M., Marsh, W., Hearty, P., Radlinski, L., & Krause, P. (2007b). Project data incorporating qualitative facts for improved software defect prediction. In Proceedings of the Third international Workshop on Predictor Models in Software Engineering, International Conference on Software Engineering (May 20 -26, 2007).
[14]. Fenton, N. E., Neil, M., & Caballero, J. G. (2007). Using ranked nodes to model qualitative judgments in Bayesian Networks. IEEE Transactions on Knowledge and Data Engineering,19(10),1420–1432. oi:10.1109/TKDE.2007.1073
[15]. Finnie, G. R., Wittig, G. E., & Desharnais, J. M. (2011). A comparison of software effort estimation techniques: Using function points with neural networks, case-based reasoning and regression models. Journal of Systems and Software, 39(3), 281–289. doi:10.1016/S0164-1212(97)00055-1
[16]. ISBSG. (2005). Estimating, Benchmarking & Research Suite Release 9. Hawthorn, Australia: International Software Benchmarking Standards Group.
[17]. Jeffries, R., Anderson, A., & Hendrickson, C. (2000). Extreme programming installed. Reading, MA: Addison-Wesley Professional.
[18]. Jensen,F. (2001).Bayesian Networks and decision graphs, New York: Springer-Verlag.
[19]. Jones, C. (1986) Programmer productivity. New York: McGraw Hill.
[20]. Jones, C. (1999). Software sizing. IEE Review, 45(4), 165–167. doi:10.1049/ir:19990406