Improved Software Defect Prediction

Martin Neil and Norman Fenton (Agena Ltd and Queen Mary, University of London)

Abstract

Although a number of approaches have been taken to quality prediction for software, none have achieved widespread applicability. This paper describes a single model 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 Bayesian Networks as the appropriate formalism for representing defect introduction, detection and removal processes throughout any life-cycle. The approach combines subjective judgements from experienced project managers and available defect rate data to produce a risk map and use this to forecast and control defect rates. Moreover, the risk map more naturally mirrors real world influences without any distracting mathematical formality. The paper focuses on the extensive validation of the approach within Philips Consumer Electronics (dozens of diverse projects across Philips internationally).

The resulting model (packaged within a commercial software tool, AgenaRisk, usable by project managers) is now being used to predict defect rates at various testing and operational phases. The results of the validation confirm that the approach is scalable, robust and more accurate that can be achieved using classical methods. We have found 95% correlation between actual and predicted defects.

The defect prediction models incorporate cutting-edge ideas and results from software metrics and process improvement research and package them as risk templates that can either be applied either off-the-shelf or after calibrating them to local conditions and to suit the software development processes in use.

1. INTRODUCTION

A number of authors, for example [3, 6, 21], have recently used Bayesian Networks (BNs) in software engineering management. In our own earlier work [8] we have shown how BNs can be used to predict the number of software defects remaining undetected after testing. This work lead to the AID tool [20] developed in partnership with Philips, and used to predict software defects in consumer electronic products. Project managers use a BN-based tool such as AID to help decide when to stop testing and release software, trading-off the time for additional testing against the likely benefit.

Rather than relying only on data from previous projects, this work uses causal models of the Project Manager's understanding, covering mechanisms such as:

- poor quality development increases the number of defects likely to be present
- high quality testing increases the proportion of defects found.

Causal models are important because they allow all the evidence to be taken into account, even when different evidence conflicts. Suppose that few defects are found during testing – does this mean that testing is poor or that development was outstanding and the software has few defects to find? Regression-based models of software defects are little help to a Project Manager who must decide between these alternatives [10]. Data from previous projects is used to build the BN, with expert judgements on the strength of each causal mechanism.

In this paper, we extend the earlier work by describing a much more flexible and general method of using BNs for defect prediction. We describe how the AgenaRisk [1] toolset is used to create an effective decision support system from the BN. We also describe a BN that models the creation and detection of software defects without commitment to a particular development lifecycle. We then show how a software development organisation can adapt this BN to their development lifecycle and metrics data with much less effort than is needed to build a tailored BN from scratch.

The contents of the remainder of the paper are as follows: in Section 2 we introduce BNs and show how they are used for causal modelling in software engineering. Section 3 introduces the idea of a 'phase' as a sub-part of a software lifecycle and shows how several phase models can be combined to model different lifecycles. The phase model is described in detail in Section 4; Section 5 shows how it is adapted to different development lifecycles. An experimental validation of defect predictions is described in Section 6.

2. DEFECT PREDICTION WITH BNs

2.1 Bayesian Networks

A Bayesian network [13] (BN) is a graph (such as that shown in Figure 1) together with an associated set of probability tables. The nodes represent uncertain variables and the arcs represent the causal/relevance relationships between the variables.

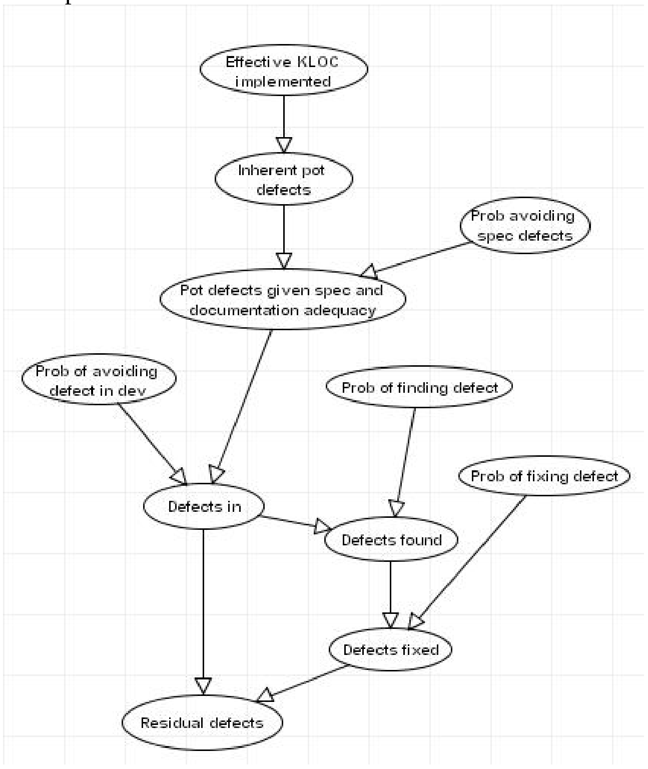
Effective KLOC implemented
Inherent pot defects
Prob avoiding spec defects
Pot defects given spec and documentation adequacy
Prob of avoiding defect in dev
Prob of finding defect
Prob of fixing defect
Defects in
Defects found
Defects fixed
Residual defects

Figure 1 BN for Defect Prediction

The BN of Figure 1 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 quality of the development and testing processes is represented in the BN of Figure 1 by four variables:

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

The BN in Figure 1 is a simplified version the BN at the heart of the decision support system for software defects. None of these probability variables (or the 'Effective KLOC implemented' variable) are entered directly by the user: instead, these variables have further parents modelling the causes of process quality as we describe in Section 4.

2.2 Building the BN Model

Like all BNs, the defect model was built using a mixture of data and expert judgements. Understanding cause and effect is a basic form of human knowledge, underlying our decisions. For example, a project manager knows that more rigorous testing increases the number – and proportion of – defects found during testing and therefore reduces the number remaining in the delivered software.

It is obvious that the relationship is not the other way round. However, it is equally obvious that we need to take into account whatever evidence we have about: the likely number of defects in the software following development; the capabilities of the team; and the adequacy of the time allowed. The expert's understanding of cause and effect is used to connect the variables of the net with arcs drawn from cause to effect.

To ensure that our model is consistent with these empirical findings, the probability tables in the net are constructed using data, whenever it is available. However, when there is missing data, or the data does not take account of all the causal influences, expert judgement must be used as well.

3. MODELLING 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.

3.1 A Lifecycle Phase

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 Section 4 we describe the BN model for one phase in more detail. First, in the next section, we describe how multiple instances of the BN are linked to model an arbitrary lifecycle.

3.2 Linking Phases: Dynamic BNs

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 2 show how this is applied when the quality attribute is the number of residual defects.

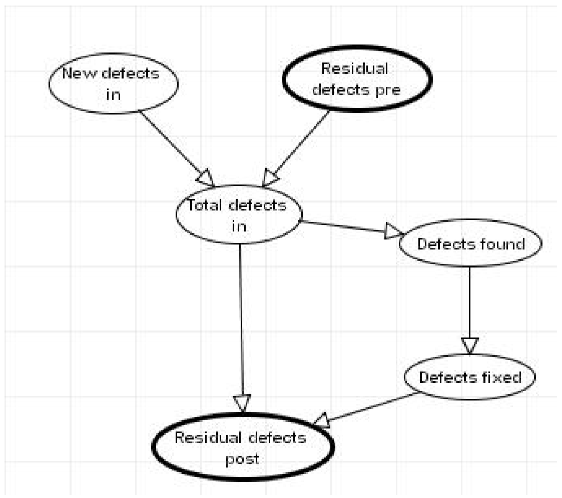

Figure 2 A Dynamic BN Modelling a Software Lifecycle

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. MODELLING A SINGLE LIFECYCLE PHASE

The phase BN is best presented as six 'subnets', each consisting of BN variables, as shown in Figure 3, with grey arrows marking where there is at least one variable in one subnet which is a parent of a variables in different subnets. The subnet plays no part in inference but is a useful for guide for the user of the BN.

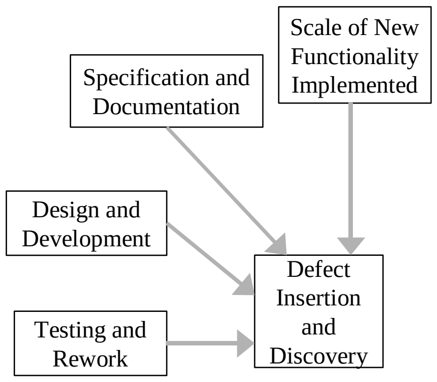
Scale of New Functionality Implemented
Specification and Documentation
Design and Development
Defect Insertion and Discovery
Testing and Rework

Figure 3 Subnets of the Phase BN

The subnets are:

- Scale of New Functionality Implemented. Since we are to build and test some software we may be implementing some new functionality in this phase. This subnet provides a measure of the size of this functionality.

- Specification and Documentation. This subnet is concerned with measuring the amount of specification and documentation work in the phase, the quality of the specification process and determining the change in the quality of the documentation as a result of the work done in the phase (modelled as a time-indexed variable).

- Design and Development. This subnet models the quality of the design and development process, which influences the probability of inserting each of the potential defects into the software.

- Testing and Rework. This subnet models the quality of the testing process and the rework process, influencing the probabilities of finding and fixing defects.

- Defect and Insertion and Discovery. This subnet follows the pattern already described in Section 2.1, adapted to handle changes to the number of defects using a time-indexed variable. 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 ‘inherent potential defects’, which are independent of the specification. The number of these is a function of the number of function points delivered (based on empirical data by Jones [14, 15]).

Figure 4 shows the testing and rework subnet.

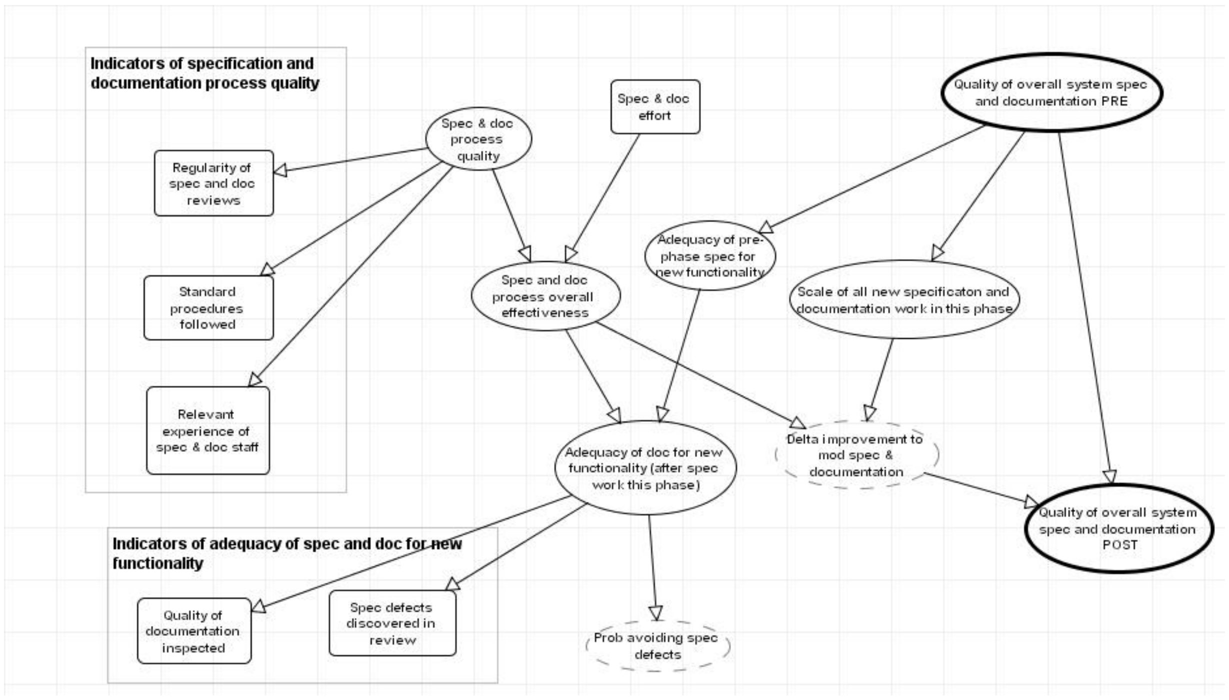
Indicators of specification and documentation process quality
Regularity of spec and doc reviews
Standard procedures followed
Relevant experience of spec & doc staff
Spec & doc process quality
Spec & doc effort
Spec and doc process overall effectiveness
Quality of overall system spec and documentation PRE
Adequacy of pre-phase spec for new functionality
Scale of all new specificaton and documentation work in this phase
Delta improvement to mod spec & documentation
Adequacy of doc for new functionality (after spec work this phase)
Quality of overall system spec and documentation POST
Indicators of adequacy of spec and doc for new functionality
Quality of documentation inspected
Spec defects discovered in review
Prob avoiding spec defects

Figure 4 Specification and Documentation Subnet

Figure 5 shows the testing and rework subnet.

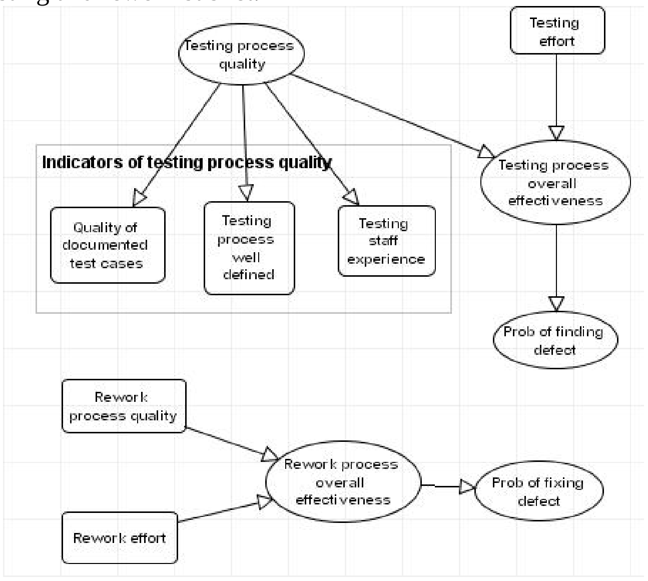
Testing process quality
Testing effort
Indicators of testing process quality
Quality of documented test cases
Testing process well defined
Testing staff experience
Testing process overall effectiveness
Prob of finding defect
Rework process quality
Rework effort
Rework process overall effectiveness
Prob of fixing defect

Figure 5 Testing and Rework Subnet

The better testing process the more likely we are to find 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 two probabilities are used to update the number of residual defects in the 'Defect Insertion and Discovery' subnet and to predict the number of residual defects at the start of any subsequent phase in which further development and/or testing of the software takes place.

5. APPLICATION METHODOLOGY

5.1 Quality Indicators

Indicator variables used in the BN can be customised to match the indicators used within the organisation. As well as editing names given to an indicator in the questionnaire, its probability table can be adjusted. The formula language of the AgenaRisk toolset makes this feasible. Consider, for example, the ‘Testing Process Quality’ (TPQ) shown in Figure 5. The suggested indicators are:

- Quality of documented test cases
- Testing process well defined
- Testing staff experienced

The process quality and the indicator values are judged on a five-point scale from ‘very low’ to ‘very high’: the judgement being relative to the norm for the development environment. To set up the indicators, an expert need only judge its ‘strength’ as an indicator of the underlying quality attribute. Given that the process quality really is high, how certain is it that the staff will be experienced? We have found the truncated normal distribution useful for creating a probability expressing an expert’s assessment of the ‘strength’ of an indicator. For example, suppose:

Testing process well defined = TNormal(‘TPQ’, 0.6)

Testing staff experience = TNormal(‘TPQ’, 0.2)

this expresses the judgement that the staff experience is the stronger indicator, since it has a smaller variance parameter (0.2) than the other indicator. In both cases the mean value of the indicator is given by the parent process quality.

5.2 Toolset

Our experience from earlier commercial projects is that project managers and other users who are not BN experts do not wish to use a BN directly via a general purpose BN editor. Instead, the BN needs to be hidden behind a more specialised user interface. The toolset provided by AgenaRisk is actually an application generator that enables toolset users to tailor both the underlying BN models and the user interface that is provided to the end-users when the application is generated.

The main functions provided to the end-user are:

- **Questionnaire interface** Observations can be entered using a questionnaire interface, where questions correspond to BN variables. Each question includes an explanation and the user can select a state (if the number of states is small) or enter a number (if the states of the variable are intervals in a numeric range). Answers given are collected into ‘scenarios’ that can be named and saved. At least one scenario is created for each software development project but it is possible to create and compare multiple scenarios for a project.

- **Output graphs** Predictions are displayed as probability distributions and as summary statistics (mean, median, and variance). Distributions are displayed either as bar charts or as line graphs (see Figure 6) depending on the type of variable and the number of states. The predictions for several scenarios can be superimposed for ease of comparison. Summary statistics can be exported to a spreadsheet.

The questionnaires shown to the end user can be configured widely. For example, questions can be grouped and ordered arbitrarily and the question text is fully editable. Not all variables need have a question, allowing any BN variable to be hidden from the end user.

An example probability distribution for defects inserted is shown in Figure 6.

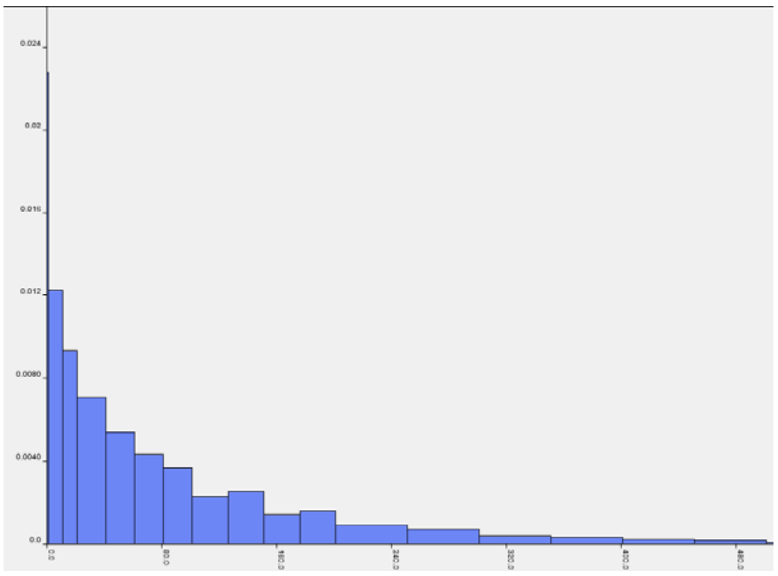

Figure 6 Predicted inserted defects distribution for a single project

6. VALIDATION

The toolset and models have been widely trialled by various commercial organisations, including Philips, Israel Aircraft Industries (Tel Aviv) and QinetiQ (Malvern). In addition, Philips has recently completed a retrospective trial of thirty-two projects carried out in Bangalore.

6.1 Aim and Methodology

The aim of the recent Philips trial was to evaluate the accuracy of the AgenaRisk defect prediction capabilities in software projects. Initially, 116 consumer electronics software projects completed between 2001 and 2004 were assessed for inclusion in the trial against the following criteria:

- reliable data was available
- project resulted in a commercial product
- some key people from the project were still available for interview

The projects should represent a range of product domains and a variety of design layers, including user interface, intermediate and driver layers.

Thirty-two projects were identified as suitable for the trial, based on these criteria.

A questionnaire, based on the AgenaRisk form for entering observations, was used to collect qualitative and quantitative information from key project members. This data was entered into the model to predict the number of defects found in testing. These predictions were then compared with the actual number of defects found in all testing phases. Data was collected in two rounds: in the second round a more detailed interview was conducted with the 'Quality Leaders' for each project resulting in improved data and improved predictions.

The trial used a variant of the 'all-activities' phase-level net. The single BN was used since it was not possible in this retrospective trial to separate the defect data into separate phases.

6.2 Results

The main results of the trial were very good indeed and a significant improvement over other approaches:

- 0-30% inaccuracy in predictions achieved on 65% of projects;
- 30-70% inaccuracy on 32% of projects
- 70% inaccuracy on 3% of projects

Figure 7 shows the results of the validation trials at Philips in the form of defects predicted and actual defects experienced. The linear correlation coefficient is 95%.

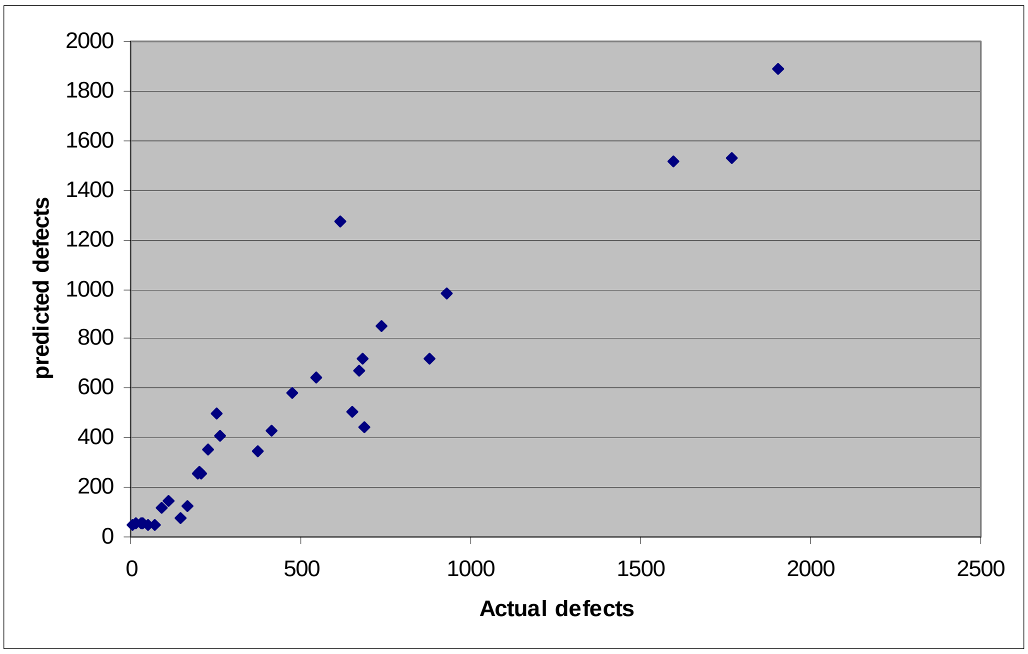

Figure 7 Scatterplot showing correlation between actual and predicted defects (95%)

7. CONCLUSIONS

We have shown how a software lifecycle can be modelled using a Bayesian Network, in which each time frame is a lifecycle 'phase' combining all software development activities in different amounts. This approach allows a BN for software defect prediction to be tailored to different software development environments. The AgenaRisk toolset makes this a practical approach, providing a formula language with standard statistical distributions which can be used to change the quality indicators available in each software development team.

The approach and toolset have been extensively trialled by industrial partners in a collaborative project. Despite making little use of the available tailoring capabilities, a retrospective trial of 32 projects showed a good fit between predicted and actual defect counts.

We have also used AgenaRisk to reason about software projects as a whole [9] and the trade-off between time, resources and quality. Many of the factors are common in these two models, covering both the assessment of process quality and the product quality achieved and required. In future, we hope to combine the two models into a single decision support system for software projects.

REFERENCES

[1] AgenaRisk: Adavanced risk analysis for important decisions. http://www.agenarisk.com

[2] Bangsø, O. and Wuillemin, P. H., Top-down construction and repetitive structures representation in Bayesian networks, In ‘Proceedings of The Thirteenth International Florida Artificial Intelligence Research Symposium Conference’, Florida, USA., 2000. AAAI Press.

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

[4] Dean, T. and Kanazawa, K. A model for reasoning about persistence and causation, Computational Intelligence, 5:142-150, 1989.

[5] eXdecide: Quantified Risk Assessment and Decision Support for Agile Software Projects, EPSRC project EP/C005406/1, www.dcs.qmul.ac.uk/~norman/radarweb/core_pages/projects.html

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

[7] Fenton, N. E., Krause, P., Neil, M., Probabilistic Modelling for Software Quality Control, Journal of Applied Non-Classical Logics 12(2), 173-188, 2002

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

[9] Fenton, N. E., Marsh, W., Neil, M., Cates, P., Forey, S. and Tailor, T. Making Resource Decisions for Software Projects. In Proceedings of 26th International Conference on Software Engineering (ICSE 2004), (Edinburgh, United Kingdom, May 2004) IEEE Computer Society 2004, ISBN 0-7695-2163-0, 397-406

[10] Fenton, N. E. and Neil, M. A Critique of Software Defect Prediction Models, IEEE Transactions on Software Engineering, 25(5), 675-689, 1999.

[11] Fenton, N. E. and Neil, M. SCULLY: Scaling up Bayesian Nets for Software Risk Assessment, Queen Mary University of London, www.dcs.qmul.ac.uk/research/radar/Projects, 2001.

[12] Fenton, N. E. and Pfleeger, S.L. Software Metrics: A Rigorous and Practical Approach (2nd Edition), PWS, ISBN: 0534-95429-1, 1998.

[13] Jensen, F.V. An Introduction to Bayesian Networks, UCL Press, 1996.

[14] Jones, C. Programmer Productivity, McGraw Hill, 1986.

[15] Jones, C. Software sizing, IEE Review 45(4), 165-167, 1999.

[16] Koller, D., Lerner, U. and Angelov, D. A General Algorithm for Approximate Inference and its Application to Hybrid Bayes Nets, In Proceedings of the 15th Annual Conference on Uncertainty in AI (UAI), Stockholm, Sweden, August 1999, pages 324—333

[17] Kozlov, A.V. and Koller, D. Nonuniform dynamic discretization in hybrid networks, Proceedings of the 13th Annual Conference on Uncertainty in AI (UAI), Providence, Rhode Island, August 1997, pages 314--325.

[18] Neil, M., Fenton, N. E., Forey, S. and Harris, R. Using Bayesian Belief Networks to Predict the Reliability of Military Vehicles, IEE Computing and Control Engineering, 12(1), 2001, pp. 11-20.

[19] Neil, M., Fenton, N. E., Nielsen, L. Building large-scale Bayesian Networks, The Knowledge Engineering Review, 15(3), 2000, pp. 257-284.

[20] Neil, M., Krause, P., Fenton, N. E., Software Quality Prediction Using Bayesian Networks in Software Engineering with Computational Intelligence, (Ed Khoshgoftaar TM), Kluwer, ISBN 1-4020-7427-1, Chapter 6, 2003

[21] Stamelosa, I., Angelisa, L., Dimoua, P., Sakellaris, P. On the use of Bayesian belief networks for the prediction of software productivity Information and Software Tech, 45 (1), 51-60, 2003.