A Case Study Validation of an Extreme Programming Bayesian Network Model

Peter Hearty, Norman Fenton, Martin Neil, David Marquez
Queen Mary, University of London
hearty, norman, martin, marquezd @dcs.qmul.ac.uk

Abstract

Bayesian networks have the ability to combine sparse data, prior assumptions and expert judgment into a single causal model. We present such a model of an Extreme Programming environment and show how it can learn from project data in order to make quantitative effort predictions and risk assessments. This is illustrated with the use of a real world industrial project.

1. Introduction

In [10] we presented the case for modeling the Extreme Programming process using Bayesian Networks (BNs). BNs allow prior assumptions, incomplete data and expert judgment to be combined in a single, causal, model, allowing us to reason in the presence of uncertainty. We now apply the model developed in [10] to a real Extreme Programming (XP) project.

The model depends heavily on *Project Velocity* (PV), the one management metric that is always available in XP. Roughly speaking, PV can be thought of as "productive effort per iteration". The exact definition of PV is given in section 2.2.

PV data is collected from the first iteration in any project. This is incorporated into the model, enabling it to learn key parameters and increase the confidence of its predictions in subsequent iterations.

As argued in [10], the model satisfies the following requirements.

1. It monitors and predicts PV, taking into account the impact of relevant process factors.
2. The core model is very small. This enables it to be replicated multiple times in order to represent the multiple iterations of an agile development environment.
3. The model can be adapted to handle different types of data for different environments. In particular, the model handles key XP practices, while being dependent on none of them.
4. Many projects report low initial productivity, gradually rising on subsequent iterations [6], [7] and [4]. As shown in [10], the model is capable of replicating this empirical behavior.
5. The model learns from data, either as a result of observations or as a result of expert judgment entered as evidence.
6. It gives useful and clear advice to managers.

While these features were initially demonstrated in [10], no attempt was made to validate the model against a real XP project. In this paper we address this by using data from a case study based around a project from Motorola [4]. The model's predictions of XP User Stories (defined below) as they are delivered over time, are in good agreement with the actual User Stories delivered.

The model was implemented using the AgenaRisk toolset [3]. This was due, amongst other things, to the ease with which dynamic models can be constructed and the availability of a wide range of built-in conditional probability functions.

Section 2 defines XP terminology as it is used in the model. Sections 3 and 4 summarize the key components of the model [10], with Bayesian Nets being discussed in Section 3 and a breakdown of the model components being presented in Section 4. Section 5 applies the model to the industrial case study. Finally, Section 6 provides some conclusions and indications of future work.

2. Definitions and Terminology

The basic unit of work in Extreme Programming (XP) is the *User Story*. When an XP iteration finishes, the estimated efforts for the completed user stories are added together to create the Project Velocity (PV). In the sub-sections that follow we describe how user stories and PV are defined, and how they are incorporated into the model

2.1. User Stories

Developers assign the effort that they believe is required for them to design, code and test each user story. Efforts are estimated using a unit called *Ideal Engineering Days* (IEDs). This is a day devoted entirely to user story completion, free from overheads and distractions. It includes detailed design, coding, unit testing and acceptance testing. It excludes all other effort that can consume developers' time, including but not

limited to administrative tasks, mentoring, support and learning.

We denote the estimated effort for the j^{th} user story in iteration i by U_i^j.

2.2. Project Velocity

Once iteration i is complete, the estimates for the completed user stories are added together. This is the project velocity V_i for iteration i.

$$V_i = \sum_{j \text{ completed in } i} U_i^j \qquad \text{Eq. 1}$$

Assuming that the next iteration, $i + 1$, is the same length, the customer selects the highest priority uncompleted user stories whose estimated IEDs sum to V_i. These user stories are then scheduled for iteration $i + 1$. The work scheduled for iteration $i + 1$ therefore has the same estimated ideal effort as the estimates for the actual work completed in iteration i.

Note that the actual time taken to complete a user story is not used here. To relate actual productive time to estimated productive time, we introduce a bias, b_i, into the model. The word "bias" is not intended in the statistical sense of a biased estimator.

If A_i^j are the actual efforts taken then:

$$b_i = \frac{\sum_j U_i^j}{\sum_j A_i^j} = \frac{V_1}{\sum_j A_i^j} \qquad \textbf{Eq. 2}$$

This scheduling mechanism assumes that the ratio of effort (people × working days) to V remains constant. This assumption can be justified by examining the two possible estimation scenarios.

1. User story estimates are being consistently overestimated or underestimated. This consistency ensures that any bias in the estimates for the previous iteration will be repeated in the current iteration.
2. There is no consistent bias in the effort estimations, i.e. there is as much overestimation as underestimation. These inaccuracies will even themselves out over multiple iterations. This further assumes that teams are able to schedule additional work in an iteration when the effort of existing tasks has been over-estimated and slack time is available.

2.3. Process factors

To model the relationship between total effort and productive effort, there is a single controlling factor which we call Process Effectiveness, e. This is a real number in the range [0,1]. A Process Effectiveness of one means that all available effort becomes part of the productive effort.

The Process Effectiveness is, in turn, controlled by two further parameters: Effectiveness Limit, l, and Process Improvement, r. The Process Improvement is the amount by which the Process Effectiveness increases from one XP iteration to the next. To allow for failing projects, the Process Improvement can take on negative values.

The Effectiveness Limit recognizes the fact that there are often limits to how productive a team of people can be. Effectiveness Limit is therefore the maximum value which the model allows Process Effectiveness to take.

3. Bayesian Net Model

A Bayesian Net (BN) is a directed acyclic graph (such as the one shown in Fig. 1), where the nodes represent random variables and the directed arcs define causal influences or functional relationships. Nodes without parents are defined through their prior probability distributions. Nodes with parents are defined through Conditional Probability Distributions (CPDs). Some CPDs are deterministic functions, such as the ones shown in Fig. 1; others, such as the priors for the initial settings in section 4.2, are defined as probability functions.

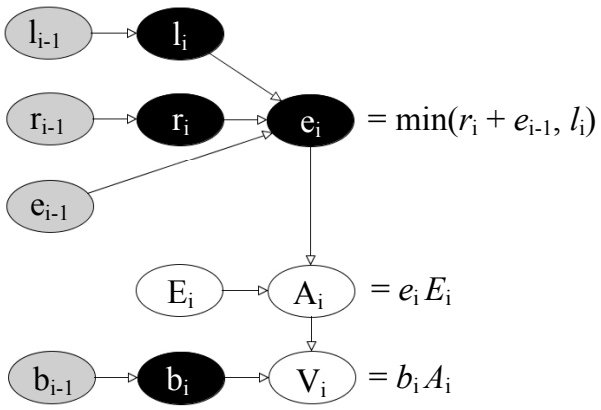

Fig. 1. Project velocity model.

Table 1 summarizes the model variables for the BN described in [10]. Measures of effort are denoted by capital letters. All other variables use lower case letters. Subscripts are used to denote a specific XP iteration. For example V_2 denotes the velocity in iteration 2. Where the iteration is not important, we drop the subscript and refer simply to V.

Table 1 Symbol definitions

Symbol	Meaning
d_i	Number of working days in iteration *i*. $d_i = 0, 1, 2, ...$ This is an integer value.
p_i	Number of team members in iteration *i*. This can be fractional if one or more people do not work full time on the project. $e_i \in [0, \infty)$.
s_i	Productive effort to date. $s_i = s_{i-1} + V_i = \Sigma V_i$, $s_i \in [0,\infty)$.
E_i	Iteration effort in man-days. $E_i = p_i \times d_i$, $E_i \in [0,\infty)$.
U_i^j	Estimated effort of j^{th} user story in iteration *i*. $U_i^j \in [0,\infty)$.
A_i	Actual productive effort in iteration *i*. $A_i = E_i \times e_i$, $A_i \in [0,\infty)$.
V_i	Project Velocity in iteration *i*. $V_i = \sum_j U_i^j$, $V_i \in [0,\infty)$.
b_i	Estimation bias. $b_i = V_i / A_i$, $b_i \in [0,\infty)$.
f_i	Load Factor in iteration *i*. $f_i = E_i / V_i$, $f_i \in [1,5]$. Used to estimate timescales. The upper limit is arbitrary.
e_i	Process effectiveness in iteration *i*. $V_i = E_i \times e_i$, $e_i \in [0,1]$.
l_i	Effectiveness limit. The maximum value that the e_i can take, $l_i \in [0,1]$.
r_i	Process improvement. $e_i = \min(e_{i-1} + r_i , l_i)$, $r_i \in [-1,1]$.

When the value of a variable has been measured, it can be entered as data in the corresponding node. The rules of Bayesian probability are then applied to propagate consistently the impact of the evidence on the probabilities of the variables of interest. More information on Bayesian Nets and suitable propagation algorithms can be found in [1] and [5].

When we wish to distinguish between a model prediction and a measured value, we will use an underscore to denote the measurement. So if V_3 is the predicted value for the velocity at iteration three, then $\underline{V_3}$ is the measured value.

Not all of the variables shown in Table 1 are shown in Fig. 1. Several of the variables are included only to make the definitions of others more rigorous (*d*, and *p*). Some exist to relate the model to XP concepts (*f* and *U*), and others to relate the model to management concepts (*s*).

Before presenting the model in detail, we need to discuss a few preliminaries about Dynamic Bayesian Nets.

3.1. Dynamic Bayesian Networks

Dynamic Bayesian Nets (DBN) extend BNs by adding a temporal dimension to the model. Formally, a DBN is a temporal model representing a dynamic system, i.e. it is the system being modeled which is changing over time, not the structure of the network [8]. A DBN consists of a sequence of identical Bayesian Nets, $\mathbf{Z}_t$, t = 1,2,..., where each $\mathbf{Z}_t$ represents a snapshot of the process being modeled at time *t*. We refer to each $\mathbf{Z}_t$ as a *timeslice*. For XP, where the software production process is split into a series of discrete iterations, this is a particularly apt approach.

The models presented here are all first order Markov. This means that the $P(\mathbf{Z}_t \mid \mathbf{Z}_{1:t-1}) = P(\mathbf{Z}_t \mid \mathbf{Z}_{t-1})$ (informally, the future is independent of the past given the present). The first order Markov property reduces the number of dependencies, making it computationally feasible to construct models with larger numbers of timeslices. Consistent propagation is achieved using standard Junction Tree algorithms [5].

Nodes that contain links between two timeslices are referred to as *link nodes*. Fig. 1 shows a single timeslice $\mathbf{Z}_t$, t =1,2…., but with the link nodes from the previous timeslice shown lightly shaded. The link nodes to the next timeslice are shaded black. Fig. 2 shows the same model, this time "rolled out" as a three iteration DBN (link nodes are shaded).

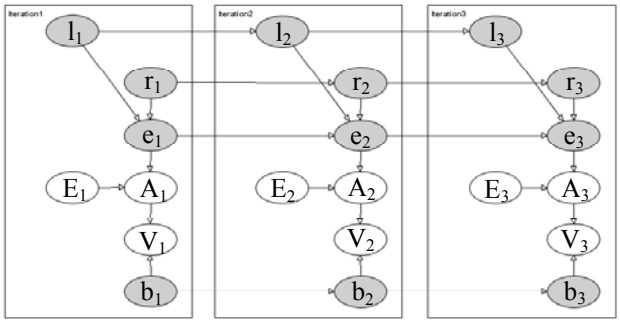

Fig. 2 Model as a DBN

3.2. Parameter Learning

The process effectiveness limit (l_i) , rate of process improvement (r_i) and bias (b_i) are the key parameters in this model. Between them they control the process effectiveness, which in turn controls the velocity. It is important that the model is capable of adjusting these parameters as a result of entering data about the project.

In particular, the model must respond to observations of the $\underline{V}_i$.

4. Model Breakdown

We describe a single timeslice by breaking it down into distinct fragments (section 4.1). The model is constructed by linking these timeslices together. A special initial timeslice is used to create the initial prior probability distributions (section 4.2).

4.1. Iteration Model

The BN shown in Fig. 1 is used as a single iteration model for project velocity. The model is best thought of as comprising three distinct fragments.

Fragment 1 controls the Productive Effort (Fig. 3). A single variable, Process Effectiveness (e_i), is assumed to determine the Productive Effort. High Process Effectiveness means a high Productive Effort and a correspondingly high velocity. Process Effectiveness increases or decreases based on the value of the Process Improvement (r_i). It is constrained to the range [0, l_i].

The CPD of l_i is a function of l_{i-1}. In this case l_i is set equal to l_{i-1}. The process effectiveness limit (l_i) is really a single variable which is global to all timeslices. Copying it between timeslices allows us to preserve the first order Markov property. Similarly r_i is just a copy of r_{i-1}.

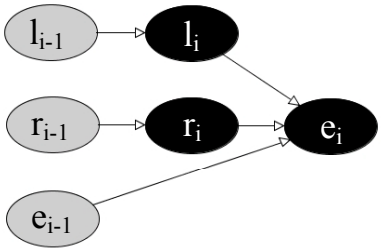

Fig. 3 Fragment 1 - Process effectiveness nodes

Fragment 2 contains the "effort" nodes (Fig. 4). It combines the total Iteration Effort (E_i) with the process effectiveness (e_i) to create the actual Productive Effort (A_i). Note that, although A_i is not required by the XP methodology, we need it in this model for reasons that will be explained below.

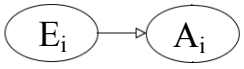

Fig. 4 Fragment 2 - Effort nodes

Fragment 3 holds the project velocity (Fig. 5). Velocity can either be predicted by the model (V_i), or once an iteration is completed, it can be entered as evidence ($\underline{V}_i$) and used to learn the model parameters. The bias, b_i, allows for any consistent bias in the team's effort estimation. If there was no bias then the productive effort, A, would be the same as V and there would be no need to distinguish between the two.

Fig. 5 Fragment 3 - Project Velocity

4.2. Setting the initial conditions

An initial timeslice, Iteration 0 (shown in Fig. 6), is used to set the initial model conditions.

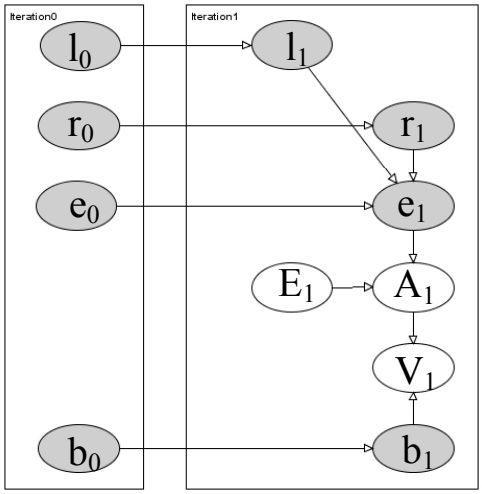

Fig. 6. Initial Velocity model

For iteration 0, the prior distributions of the input effectiveness limit (l_0), process improvement (r_0) and process effectiveness (e_0) are all set to be normal distributions, with variances of 0.3 and means of 0.8, 0.2 and 0.3 respectively. These values are based on a controlled case study by Abrahamsson and Koskela [7], where process effectiveness varied between 0.4 and 0.75. We have simply extended this range slightly and chosen r_0 so that the lowest to highest transition can take place within four iterations.

The prior of the estimation bias (b_0) is set to a log normal distribution with a mean of approximately 1.0, and a variance of 0.1. The log normal distribution follows from the fact that the bias cannot be less than zero but has no upper bound. For example, a pessimistic bias, where estimates are 2 times the actual, results in a bias of 2, whereas an optimistic bias results in a bias of ½. This distribution is confirmed empirically, for example by Little [12].

Evidence is entered in all of the E_i nodes.

5. Model Validation

In [10] we showed that the model reproduced known empirical behavior in XP projects. This includes the gradually increasing values of V observed in several studies [6], [7] and [4].

In this section we apply the model to an industrial case study (section 5.1). The model can learn from the initial data entered from the project (section 5.2) and adjusts its predictions once beneficial XP practices are taken into account (section 5.3) Section 5.4 provides an example of how the model can be calibrated for a specific XP practice. Finally, in Section 5.5 the model provides predictions for the time taken to deliver a fixed amount of functionality. These are in good agreement with the actual functionality delivered.

5.1. The Motorola Project

Williams, Shukla and Anton [4] provided a detailed description of an XP project developed at Motorola. The project was developed in a series of eight iterations of between two and three weeks duration. The number of people on the team varied from three to nine people over the duration of the project. The full data set is shown in Table 2.

Table 2 – Motorola project data

i	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*
d_i	15	15	15	16	12	10	8	10
p_i	3	3	6	6	7	7	9	4
E_i	45	45	90	96	84	70	72	40
$\underline{V}_i$	9	13	35	30	40	40	36	20

The definition of Project Velocity used by the Motorola team corresponds to what we have called Process Effectiveness. We will continue to use the definition given in Eq. 1. The values for $\underline{V}_i$ given in Table 2 have been calculated using our definition.

Initially we simply enter values for E_i into the model (no values for $\underline{V}_i$ entered). Fig. 7 shows the resulting marginal distributions which are generated for the V_i node. There is one distribution for the node in each timeslice.

The median values from the V_i distributions are shown in Fig. 8 (the "Predicted" graph). Actual values for $\underline{V}_i$ are shown in the same figure for comparison (the "Actual" graph).

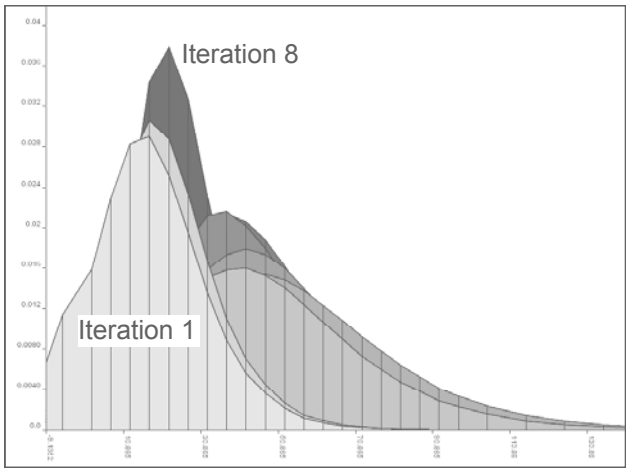

Fig. 7 Distributions for V_i, one per timeslice

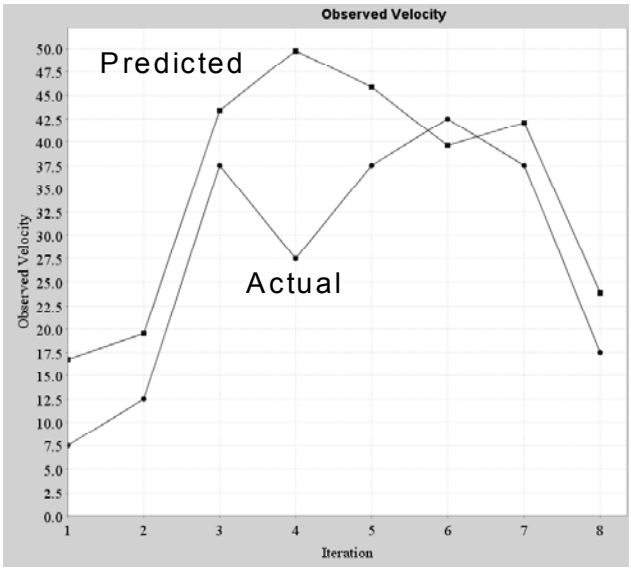

Fig. 8. Predicted vs. actual Motorola V (medians)

5.2. Parameter Learning

There are a number of problems with the predicted values in Fig. 8. The most obvious is that, apart from iteration 6, the predicted values are consistently too high. In this section we demonstrate how the model can learn from real project data and quickly improve the accuracy of its predictions.

A good example of this can be seen in Fig. 9. This shows the behavior of the Bias node, b_i, when all of the $\underline{V}_i$ values have been entered. The central dotted line, which is almost co-incident with the solid line, shows the mean and median values respectively. The outer dotted lines

show the mean ± 1 standard deviation (SD). The SD gets smaller as more evidence is entered into the model. This illustrates that, not only does the model learn the values of its parameters, but the uncertainty in those values decreases as more evidence becomes available.

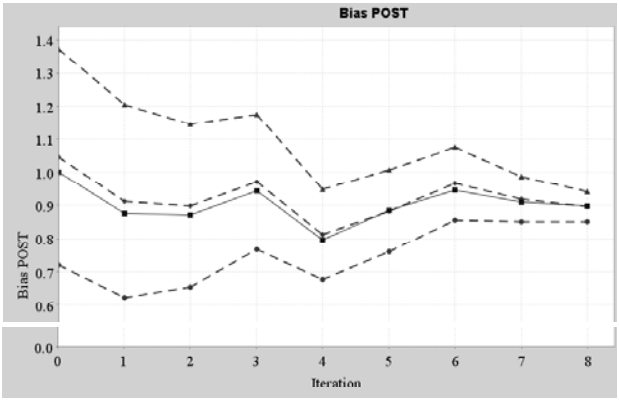

Fig. 9 Bias - median, mean ± 1 SD, actual data

The effect of this learning process can be seen by taking the "Predicted" scenario and entering $\underline{V_i}$ observations for completed iterations. As each new piece of information is entered, back propagation takes place, causing the distributions for the model parameters to be updated. These updated parameter distributions then affect the predictions of future iterations.

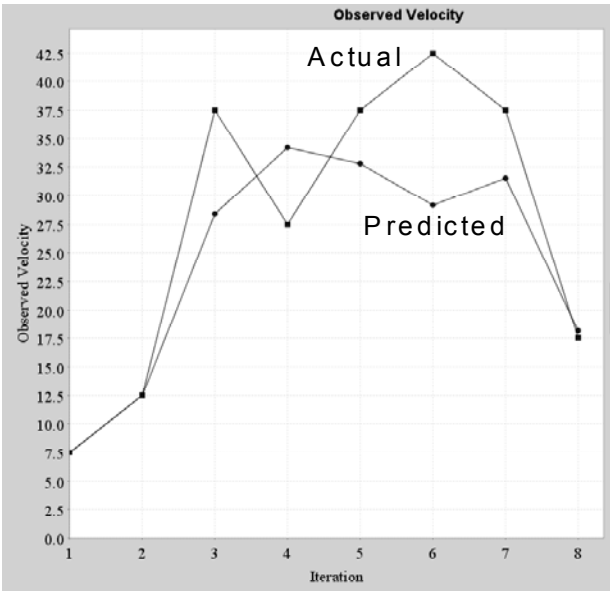

Fig. 10 Predicted vs. actual *V*, 2 observations

The graphs in Fig. 10 show the change in predicted values when the first two $\underline{V_i}$ values have been entered. The whole of the "Predicted" graph moves to lower values as the model learns from the observations. The predictions for V_i in iterations 3 and 4 improve as a result. However, the predicted V_i for iterations 5, 6 and 7 are significantly worse.

The Williams, Shukla and Anton paper [4] points out that various XP practices were implemented more effectively in later iterations. In the next section, we show how this can be incorporated into the model.

5.3. Indicator Nodes

Indicator nodes are nodes with a single parent and no children. They are often used to provide evidence for variables that are themselves unobservable. Indicator nodes are one of the main mechanisms used to introduce XP practices into the model.

An indicator node for the Effectiveness Limit is shown in Fig. 11: the "Onsite Customer" node. This is the extent to which an authoritative customer was available to answer questions about requirements and provide feedback on development. It is a ranked node, consisting of five discrete values ranging from Very Low to Very High. These discrete values define five equal, discrete partitions of the real number range [0,1].

The probability of these five values is derived from a truncated normal distribution whose mean is l_i, and whose variance is set to 0.1. This distribution ensures that a high degree of customer input leads to a high effectiveness limit. The variance determines the strength of the relationship. More information on ranked nodes and the use of the truncated normal distribution can be found in [11].

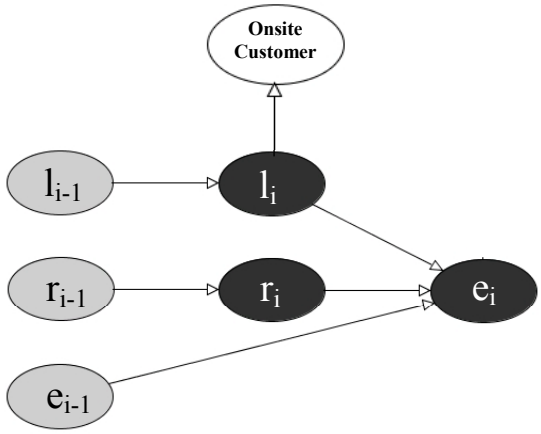

Fig. 11 The "Onsite Customer" indicator node

It is important to emphasize that the values entered into the "Onsite Customer" node must be relative to the need for customer input. If the project team have developed similar projects for this customer in the past, or are themselves experts in the application domain, then constant customer input may not be useful. In these circumstances a "Very High" value for "Onsite

Customer" might be appropriate, even if the customer is not physically present, but was still able to provide input when needed.

Fig. 12 shows how the indicator node's parent is affected by changes in its values. The central, straight line shows the median from the Effectiveness Limit node's distribution when only effort data has been entered; this is the "Predicted" scenario. When all the $\underline{V}_i$ data is entered, then the Effectiveness Limit varies throughout the project (the "Actual" curve). The "Learned" curve shows the Effectiveness Limit that is learned when only $\underline{V}_1$ and $\underline{V}_2$ have been entered as observations. This is the curve which is responsible for the modified predictions shown in Fig. 10.

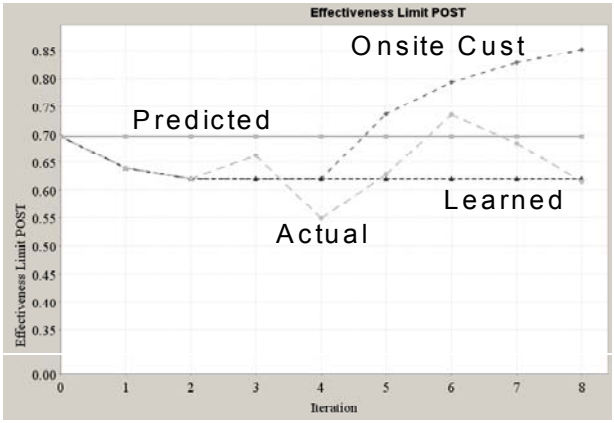

Fig. 12. Effectiveness Limit with and without indicator node evidence

At the start of the 5th iteration, the Motorola team had constant access to an onsite customer. The "Onsite Customer" indicator node was therefore set to "Very High" for these iterations. The result is the "Onsite Cust" curve. It shares the same values for the Effectiveness limit as the "Learned" curve, until the values for the Onsite Customer indicator node are modified.

The result of entering indicator node evidence is an improvement in the predicted V_i values, as shown in Fig. 13.

5.4. Calibrating the Onsite Customer Node

The distribution for the "Onsite Customer" node is based on data from Korkala, Abrahamsson and Kyllönen [9]. In their paper, four case studies are described with varying degrees of customer interaction. The percentage of effort devoted to fixing defects, including specification defects, varied greatly in the four case studies. Where customer input was very high, only 6% of effort was spent fixing defects. Moreover this level remained constant across iterations. At the other extreme, when customer input was very low, the time spent fixing defects grew across iterations until it reached about 40% in iteration 3.

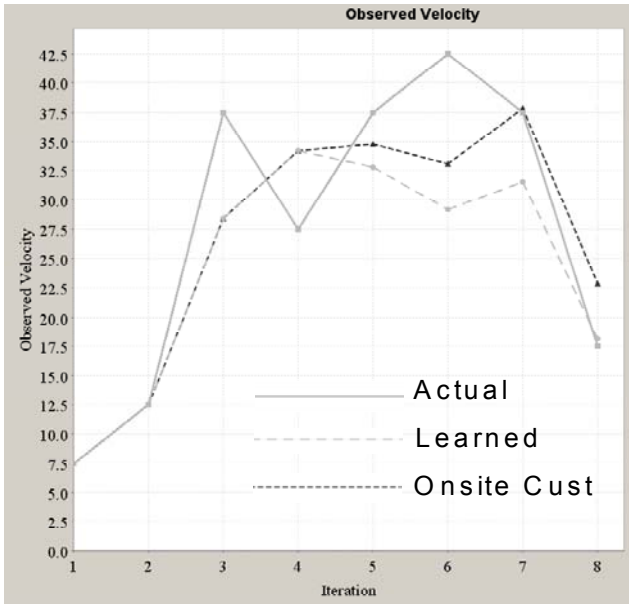

Fig. 13 *V* with and without indicator node evidence

Our model does not explicitly include details of rework effort or defect fixing effort; they are simply included as effort which does not contribute to *V*. We therefore make the following definitions and assumptions concerning the relationship between defect fixing effort and non-velocity effort.

1. Define "Misdirected Effort", m_i, to be the fraction of effort which does not contribute to completed user stories.
2. Misdirected effort is composed of effort to fix defects, d_i, and other overheads, o_i.
3. The other overheads, o_i, are fixed for a given environment.
4. When the onsite customer input is at its maximum, the defects fixing effort is at its minimum.

With these assumptions in place, we can use the Bayesian Network shown in Fig. 14 to calibrate the Onsite Customer node. The algorithm proceeds as follows.

1. An initial guess is made at the Onsite Customer distribution.
2. The values of o_i are chosen so that, when the Onsite

customer node is set to "Very High", d_i produces a constant mean value of about 6% across all iterations.

3. Modify the Onsite Customer distribution, with the value set to "Very Low" until the time spent fixing defects in iteration 3 is about 40%.
4. Repeat steps 2 and 3 until both conditions are satisfied simultaneously.

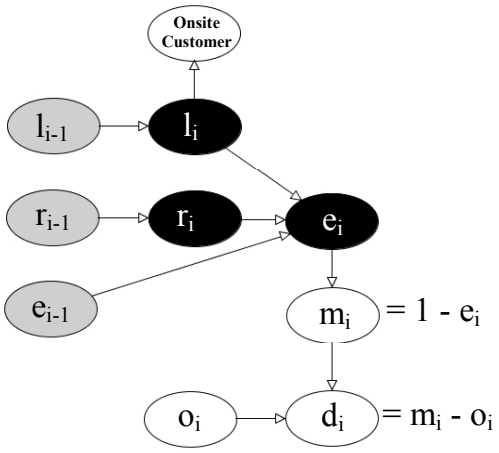

Fig. 14 BN used to calibrate the Onsite Customer node

The resulting defect effort percentages for each value of "Onsite Customer" across four iterations are shown in Fig. 15. These are similar to the empirical curves of Figure 3 in [9].

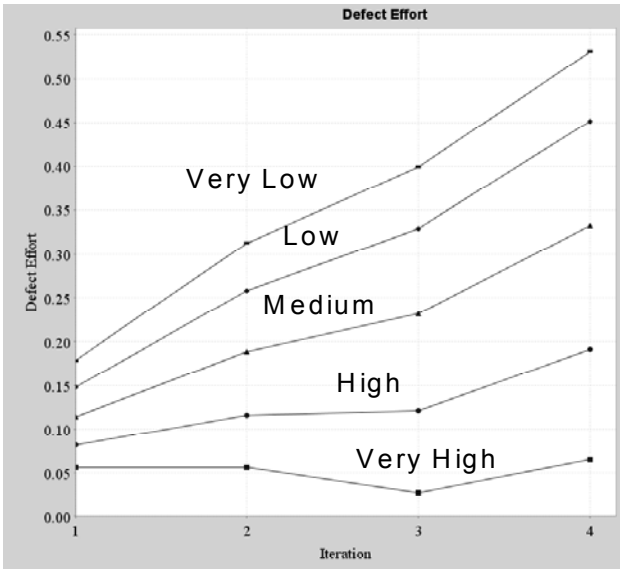

Fig. 15 Defect effort % for each Onsite Customer setting

5.5. Timescale Prediction

Fig. 16 shows a slightly modified version of the velocity fragment of the model. This includes an additional link node, s_i, which acts as the cumulative sum of V to date.

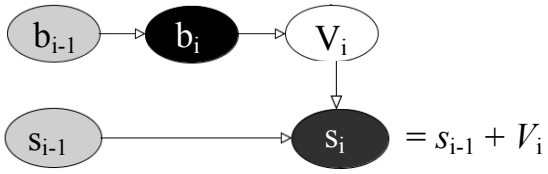

Fig. 16. Project Velocity summed to date.

Plots of s_i for the initial prediction, the learned prediction and the actual scenarios are shown in Fig. 17. If the total estimate to complete the entire project is, say, 200 IEDs, then we can immediately read off from the graph how long it will take to complete the project.

The initial predictions of the model are too optimistic. However, once the model has learned from the $\underline{V_1}$ and $\underline{V_2}$ observations, and account has been taken of the onsite customer, the predictions are virtually indistinguishable from the actual outcome.

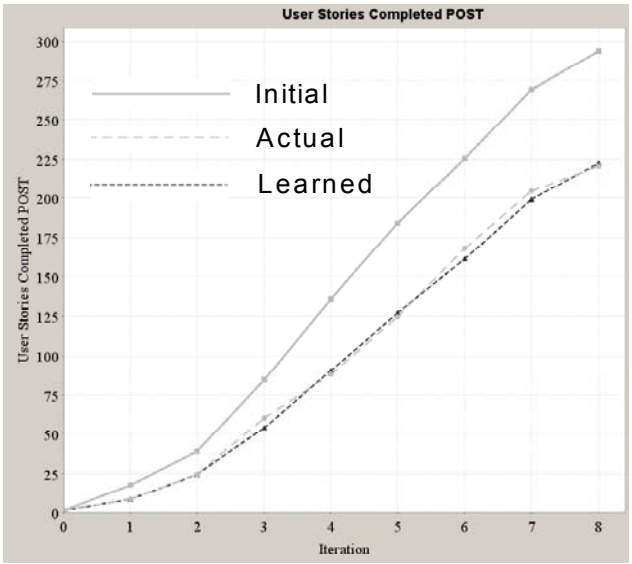

Fig. 17 Sum V_i to date

The model can also quantify the uncertainty involved in completing 200 IEDs of user stories within 8 iterations. Fig. 18 shows the cumulative distribution functions for the s_i node in iteration 8. The vertical line allows us to read off the probability of completing up to 200 IEDs by the end of the 8th iteration. For the "Initial" scenario, there is only a 10% chance of completing less than 200 IEDs, i.e. there is a 90% chance of completing 200 IEDs

or more.

Once the model has learned from $\underline{V}_1$ and $\underline{V}_2$, the probability is revised down to a 65% probability. The actual number of IEDs delivered was 224.

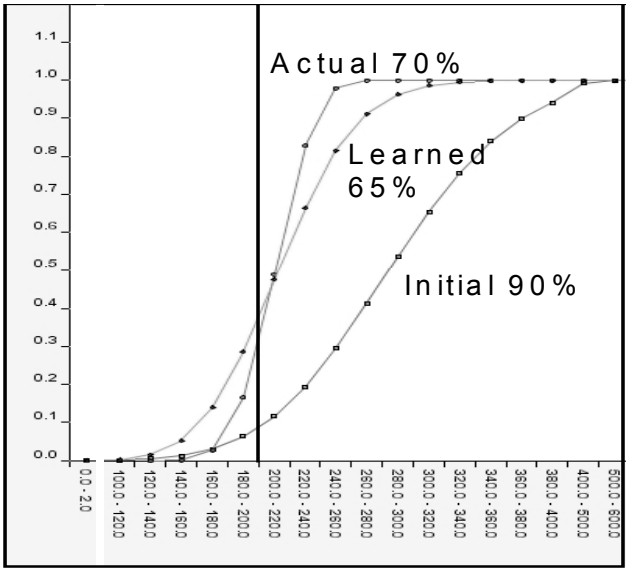

Fig. 18 Iteration 8 cumulative distributions, Sum *V* to date

6. Conclusions and Future Work

The model developed in [10] has been applied to a real industrial project. Incorporating data from the early part of the project enabled the model to update its parameters and improve its predictions. When this was combined with knowledge about the presence of an onsite customer, the model was able to make extremely accurate predictions about the level of functionality delivered over time. In [10] we showed how collective code ownership could be used in a similar role. Other XP practices can be incorporated in the model using similar techniques.

There are several things which should be noted about this type of model.

1. No extensive data collection phase was necessary. The model started off making generic predictions, but quickly altered them as local data became available.
2. Empirical data, project data, prior assumptions and expert judgment are combined in a single intuitive, causal model.
3. The predictions provide probability distributions, not just single values. The model tells you what the chances of various outcomes are.
4. It is relatively simple to add new XP practices or other environmental features, making the model extremely versatile.

The model has already been extended to reproduce the mentoring overhead of assimilating additional team members. This closely follows the model of Williams, Sukla and Anton[4]. This is an important aspect of the "Brooks' factor", i.e. the tendency for larger teams to be less productive [2].

A similar approach can be used to create a defects prediction model, with the effort model as one of its primary inputs. This allows a family of models to be constructed which represent a wide variety of XP environments and which can be used to model either effort alone, effort plus defects, or cost versus time trade-offs. Each will be able to cope with varying size project teams.

10. References

[1] Jensen, Bayesian Networks and Decision Graphs, Springer-Verlag, New York, 2001.

[2] Brooks FP, The Mythical Man-Month: essays on software engineering, 2nd edition, Addison Wesley, 1995.

[3] AgenaRisk User Manual, Agena Limited, www.agenarisk.com.

[4] Williams L, Shukla A, Antón AI, An Initial Exploration of the Relationship Between Pair Programming and Brooks' Law, Proceedings of the Agile Development Conference (ADC'04)

[5] Lauritzen, S. L. and Spiegelhalter, D. J. Local computations with probabilities on graphical structures and their application to expert systems (with discussion). J.R. Statistical Soc. Series B, 50, no. 2, pp. 157-224, 1988

[6] Ahmed, A.; Fraz, M.M.; Zahid, F.A., Some results of experimentation with extreme programming paradigm, 7th International Multi Topic Conference, INMIC 2003. Page(s): 387-390

[7] Abrahamsson P, Koskela J, Extreme Programming: A Survey of Empirical Data from a Controlled Case Study, 2004 International Symposium on Empirical Software Engineering (ISESE'04), pp. 73-82

[8] K. P. Murphy. Dynamic Bayesian Networks: Representation, Inference and Learning. PhD thesis, UC Berkeley, 2002.

[9] Korkala M, Abrahamsson P, Kyllönen P, A Case Study on the Impact of Customer Communication on Defects in Agile Software Development, Proceedings of AGILE 2006 Conference (AGILE'06)

[10] Hearty P, Fenton N, Marquez D, Neil M, Improved prediction in Extreme Programming Projects using Bayesian Networks, submitted IEEE TSE Nov 2006, http://www.dcs.qmul.ac.uk/~norman/papers/

[11] Fenton NE, Neil M "Using Ranked nodes to model qualitative judgements in Bayesian Networks" submitted IEEE TKDE, Feb 2006, http://www.dcs.qmul.ac.uk/~norman/papers/

[12] Little T, Schedule Estimation and Uncertainty Surrounding the Cone of Uncertainty, IEEE SOFTWARE May/June 2006

Peter Hearty is a Ph.D. student at Queen Mary, University of London. He gained a B.Sc. in Mathematics and Physics from the University of Stirling in 1982. He worked as a programmer, analyst and designer for various commercial organizations before founding his own database company in 1997.

Norman Fenton is Professor of Computing at Queen Mary (London University) and is also CEO of Agena, a company that specialises in risk management for critical systems. At Queen Mary he is the Computer Science Department Director of Research and he is the Head of the Risk Assessment and Decision Analysis Research Group (RADAR). Norman's books and publications on software metrics, formal methods, and risk analysis are widely known in the software engineering community. Norman's recent work has focused on causal models (Bayesian Nets) for risk assessment in a wide range of application domains such as vehicle reliability, embedded software, transport systems, TV personalisation and financial services. Norman is a Chartered Engineer and Chartered Mathematician and is a Fellow of the British Computer Society. He is a member of the Editorial Board of the Software Quality Journal.

David Marquez is a Research Assistant for the RADAR (Risk Assessment and Decision Analysis) Group, at the Department of Computer Science, Queen Mary, University of London. Before joining academia he worked as a Senior Researcher in the Oil industry, developing and applying mathematical and statistical models in reservoir characterisation problems. His research interests include Bayesian statistical modelling, Bayesian Networks, Space-State models, and statistical pattern recognition. He has a PhD in mathematic from the University of Marne-La-Valle, France.

Martin Neil is a Reader in "Systems Risk" at the Department of Computer Science, Queen Mary, University of London, where he teaches decision and risk analysis and software engineering. Martin is also a joint founder and Chief Technology Officer of Agena Ltd, who develop and distribute AgenaRisk, a software product for modelling risk and uncertainty. His interests cover Bayesian modelling and/or risk quantification in diverse areas: operational risk in finance, systems and design reliability (including software), project risk, decision support, simulation, AI and personalization, and statistical learning. Martin earned a BSc in Mathematics, a PhD in Statistics and Software Metrics and is a Chartered Engineer.