Predicting Project Velocity in XP using a Learning Dynamic Bayesian Network Model

Journal:	*Transactions on Software Engineering*
Manuscript ID:	draft
Manuscript Type:	Regular
Keywords:	D.2.18.e Software process models < D.2.18 Software Engineering Process < D.2 Software Engineering < D Software/Software Engineering, D.2.17.i Programming paradigms < D.2.17 Software Construction < D.2 Software Engineering < D Software/Software Engineering, D.2.18.c Process measurement < D.2.18 Software Engineering Process < D.2 Software Engineering < D Software/Software Engineering

Predicting Project Velocity in XP using a Learning Dynamic Bayesian Network Model

Peter Hearty, Norman Fenton, David Marquez, Martin Neil

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.**

Index Terms— **extreme programming, Bayesian nets, causal models, risk assessment**

1. INTRODUCTION

Extreme Programming (XP) is one of several iterative approaches to software development, collectively known as “Agile” methods [31]. It consists of a collection of values, principles and practices as outlined by Kent Beck, Ward Cunningham and Ron Jeffries [20] [21] [22]. These include most notably: iterative development, pair programming, collective code ownership, frequent integration, onsite customer input, unit testing, and refactoring.

XP emphasizes a lightweight, often informal approach. There are no large-scale requirements, analysis and design phases, and so there are none of the traditional metrics associated with the requirements or design phases, such as Function Points [36]. Instead, the customer and development team agree a series of User Stories (described later) that concisely define the requirements. The definition of a User Story is not as well defined as a Function Point. As such, User Stories are currently of limited value in predicting effort or quality. Yet managers of XP

Manuscript received August 9, 2006. This research is funded through eXdecide (EPSRC Grant Reference: EP/C005406/1) and an associated CASE award from Agena Ltd.
Peter Hearty and David Marquez are with Queen Mary, University of London, UK (+44 20 7882 7896, e-mail: hearty@dcs.qmul.ac.uk, marquezd@dcs.qmul.ac.uk).
Norman Fenton is with Queen Mary, University of London, UK (+44 20 7882 7860, e-mail: norman@dcs.qmul.ac.uk). He is CEO of Agena Ltd.
Martin Neil is with Queen Mary, University of London, UK (+44 20 7882 5221, e-mail: martin@dcs.qmul.ac.uk). He is CTO of Agena Ltd.

projects have just as great a need for such predictions as managers on any other software project.

Management gets some indication of the effort required based on the developers' estimates for completing user stories, but these only cover time spent working directly on the user stories. They do not cover other overheads. Managers need to know how accurate developers' estimates are and how they translate to calendar time. Only then can a project manager determine how long the project will actually take to complete. This paper addresses this problem by exploiting ideas on causal modeling that have led to improved effort and quality prediction for traditional software development.

Fenton and Neil [10] explained the rationale behind creating causal models of the software development process using Bayesian Nets (BN). BNs offer the advantage of being able to reason in the presence of uncertainty, prior assumptions and incomplete data. They can intermix expert judgment, statistical distributions and observations in a single model. Further, they are able to learn from evidence in order to update their prior beliefs. In an iterative development environment, such as XP, we can take advantage of this learning capability, where information obtained from early iterations in the project can be used to adapt a model to the local environment.

This paper presents a BN model of *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 3.2.

We set the following key requirements that the model must satisfy.

1. It must monitor and predict PV, taking into account the impact of relevant process factors.
2. For computability reasons, the core model must be 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 must be able to handle different types of data for different environments. In particular, the model must handle key XP practices, while being dependent on none of them.
4. The model must be capable of replicating empirical behavior. In particular, many projects report low initial productivity, gradually rising on subsequent iterations [6], [7] and [4].
5. The model must learn from data, either as a result of observations or as a result of expert judgment entered as evidence.
6. It must give useful and clear advice to managers.

The main contribution of this paper is to introduce and validate dynamic Bayesian nets as a means of modeling iterative software development. PV data is collected from the first iteration in any XP project. This is incorporated into the model, enabling it to learn key parameters and increase the confidence of its predictions in subsequent iterations. We show that, with very little data, it is possible to correct the model's prior assumptions and quickly produce accurate models of PV with associated risk assessments.

The remainder of this paper is organized as follows. In section 2 we discuss related models, both models of XP and other BN models of the software engineering process. Section 3 covers some of the definitions needed by the model. Section 4 presents the model itself. Section 5 covers model behavior using hypothetical data while section 6 validates the model using a real XP project. Section 7 discusses the implications of the model, including threats to its validity and some conclusions.

2. Related Work

We discuss related work under two headings. We first examine other predictive models of XP development, before going on to describe BN models of the software development process.

2.1. Extreme Programming Predictive Models

Williams and Erdogmus [18] developed a Net Present Value (NPV) [19] model of Pair Programming (PP) – one of the key practices advocated by XP. NPV models take into account the fact that earnings in the future are worth less than the same dollar earnings today. The model combines:

1. productivity rates,
2. code production rates (derived from the literature),
3. defect insertion rates
4. and defect removal rates.

Using empirical values for PP productivity and delivered defect rates [23] [24] [25], the model predicts that pair programming is a "viable alternative to individual programming".

Padberg and Müller [26] also created an NPV model of XP. Their model uses market pressure as the principle means of discounting the NPV. The model was tested under various different assumptions about performance and defect rate improvements under PP. The results indicate that the value of both of these parameters is crucial. When market pressure is high, and there is sufficient improvement in both LOC and defect rates, then PP can indeed deliver an advantage.

Several groups have constructed System Dynamics (SD) Models of XP. SD models a system as a collection of stocks, flows and feedback loops, and was first applied to software engineering by Abdel-Hamid [28]. Misic, Gevaert, and Rennie [27] attempted to model the interaction of various

XP practices. They particularly concentrated on pair programming, refactoring, test driven development and iterative development. Simulation results indicated that XP has an advantage when pairs worked well together and did not swap frequently.

Kuppuswami, Vivekanandan, and Rodrigues [29] also created an SD model. They were able to successfully simulate the flattened cost of change curve claimed by Beck [21] (p. 23).

Cau et al [32] developed a custom simulation to model the XP process, calibrated using data from a real XP project. Once calibrated, their model was able to reproduce empirically derived results [33] about the effects of test driven development (one of the recommended XP practices).

All of the above provide explanation, insight or validation of XP techniques. What none claims to do is offer combined prediction and risk assessment for project managers. This can be achieved using causal models, which are now being used effectively in traditional software engineering.

2.2. Bayesian Net Models in Software Engineering

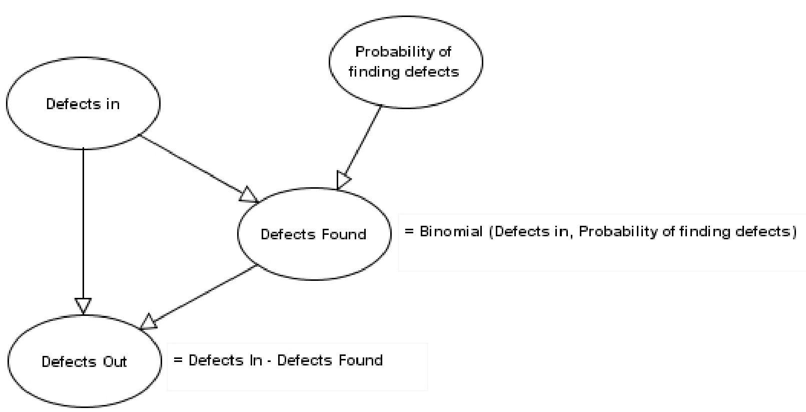

Fig. 1 An example of a Bayesian Network

A Bayesian Net (BN) [1] is a directed acyclic graph (such as the example shown in Fig. 1), where the nodes represent random variables and the directed arcs define causal influences or functional relationships. Nodes without parents (such as the “Probability of finding defects” and “Defects In” nodes in Fig. 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 Fig. 1), others (such as the "Defects Found" node in Fig. 1) are defined as standard probability distribution functions.

As explained by Fenton and Neil [10], BN models have several advantages over regression based models. BNs do not rely on point values of parameters that have been derived through some "best fit" procedure. Instead, the whole distribution of a variable is included. Similarly, BN models don't just predict a single value for a variable; they predict its probability distribution. By taking the marginal distributions of variables of interest, we get a ready-made means of providing quantitative risk assessment.

When a variable is actually observed, this observation can be entered into the model. An observation reduces the marginal probability distribution for the observed variable to a probability of 1 for the observed state (or a small interval containing the value in the continuous case) and zero otherwise. 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. More information on BNs and suitable propagation algorithms can be found in [1] and [5].

Fenton, Neil, and others have gone on to develop a series of BN models, culminating in the AID tool [13], the MODIST models [14], and the extensive trials of revised models at Philips [15]. A similar model has been developed by Siemens [34]. Those models were used to provide improved methods of risk assessment for project managers, with special emphasis on defect predictions and effort prediction.

Several other groups have also researched the use of BN based software process models. Wooff, Goldstein, and Coolen [17] have developed BNs modeling the software test process while Stamelos et al [35] used COCOMO81 cost factors to build a BN model of software project productivity. Bibi and Stamelos [16] have shown how BNs can be constructed to model IBM's Rational Unified Process.

While these models can be adapted to agile development processes, they are not specifically targeted at such environments. Agile methods, such as XP, are characterized by highly iterative approaches to software development. If each iteration is treated as if it were a mini-project in its own right, then existing models would quickly result in BNs which are unmanageable, laborious to maintain, and computationally infeasible. What is needed is an extremely small core model that can be extended as needed.

3. Definitions and Terminology

The basic unit of work in 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

3.1. 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.

3.2. 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 effort to complete a user story is not used here. To relate actual productive effort to estimated productive effort (i.e. PV), we introduce a bias, b_i, into the model. Note that the word "bias" is not intended in the statistical sense of a biased estimator.

If A_i^j are the actual efforts taken then:

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

3.3. Process factors

To model the relationship between total effort and PV, there is a single controlling factor which we call Process Effectiveness, e. Process Effectiveness 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.

Note that all of this relies on minimal assumptions: effort either contributes to delivered functionality, or it does not. The ratio between productive effort and total effort exists whether we call it Process Effectiveness or not. This ratio varies between iterations and has a limit, even if the limit is unity. As the core model contains variables based only on these factors, it too is based upon minimal assumptions.

4. Bayesian Net Model

The BN used to model project velocity is shown in Fig. 2. Table 1 summarizes the model variables for the BN. 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.

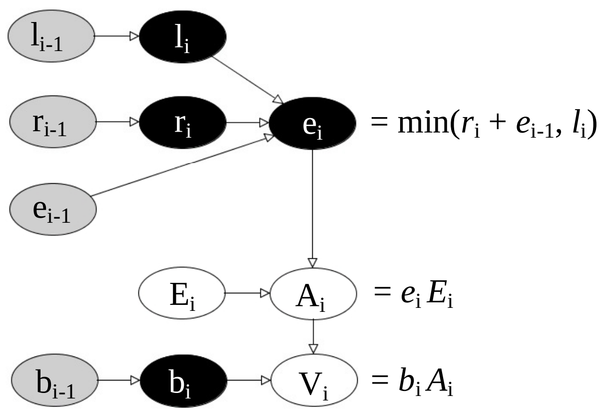

Fig. 2 Project velocity model

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.

Table 1 Symbol definitions

Symbol	Meaning
d_i	Number of working days in iteration *i*. $d_i = 0, 1, 2, \ldots$ 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. $e_i = \min(e_{i-1} + r_i , l_i)$, $r_i \in [-1,1]$.

Not all of the variables shown in Table 1 are shown in Fig. 2. 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.

4.1. 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 $P(\mathbf{Z}_t \mid \mathbf{Z}_{1:t-1}) = P(\mathbf{Z}_t \mid \mathbf{Z}_{t-1})$ (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]. Junction tree algorithms provide exact (as opposed to approximate) propagation in discrete BNs and are generally regarded as among the most efficient such algorithms [30].

Nodes that contain links between two timeslices are referred to as link nodes. Fig. 2 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. 3 shows the same model, this time "rolled out" as a three iteration DBN (link nodes are shaded).

The models in this paper were implemented using the AgenaRisk toolset [3]. This was due, amongst other things, to its ability to build dynamic models, to handle continuous variables and the availability of a wide range of built-in conditional probability functions.

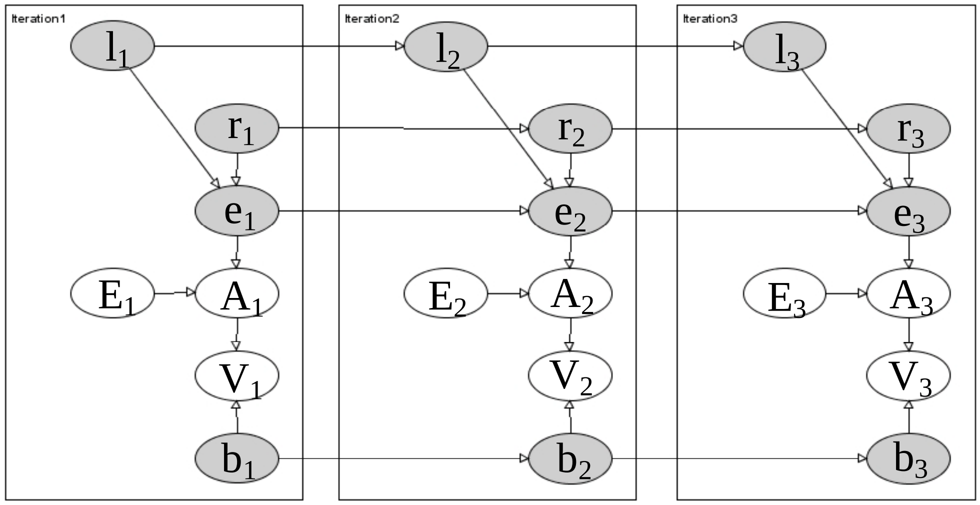

Fig. 3 Model as a DBN

4.2. 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 node, which in turn controls the velocity node. 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 $\underline{V}_i$.

4.3. Iteration Model

The BN shown in Fig. 2 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. 4). 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}.

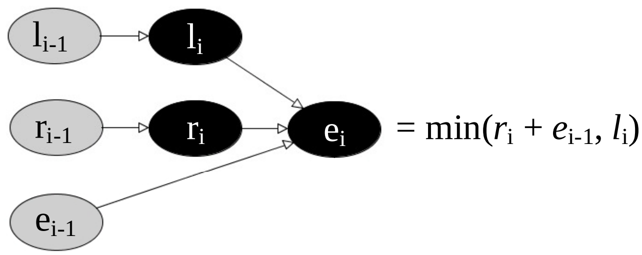

Fig. 4 Fragment 1 - Process effectiveness nodes

Fragment 2 contains the "effort" nodes (Fig. 5). 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. We do not expect A to be observed in real projects.

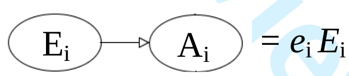

Fig. 5 Fragment 2 - Effort nodes

Fragment 3 holds the project velocity (Fig. 6). 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.

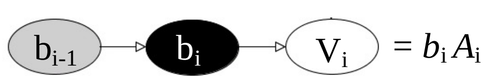

Fig. 6 Fragment 3 - Project Velocity

4.4. Setting the initial conditions

An initial timeslice, Iteration 0 (shown in Fig. 7), is used to set the initial model conditions.

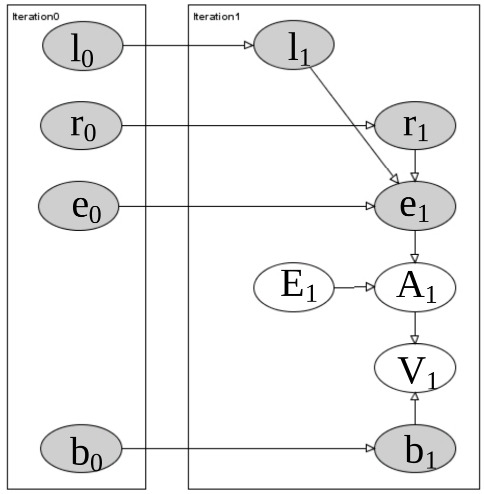

Fig. 7 Initial Velocity model

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 0.5. This distribution is confirmed empirically, for example by Little [12].

The choice of these priors is discussed further in the “Conclusions and Discussion” section of this paper.

Evidence is entered in all of the E_i nodes so the prior distributions these nodes have no effect.

5. Model Behavior

Fig. 8 shows the predicted values of the PV for a hypothetical project with 10 iterations and 50 hours of effort available in each iteration (i.e. E_i = 50, i = 1,…,10). The central dotted line is the mean, with the outer dotted lines showing +/- one standard deviation. The solid line is the median value. This is based solely on the model's initial conditions.

The Process Effectiveness increases with each iteration by an amount equal to the Process Improvement. It flattens out as it begins to hit the Effectiveness Limit. As can be see from the graph, this leads to the PV starting fairly low and gradually increasing with each iteration. Being able to model and predict this type of behavior was one of the main objectives of the core model.

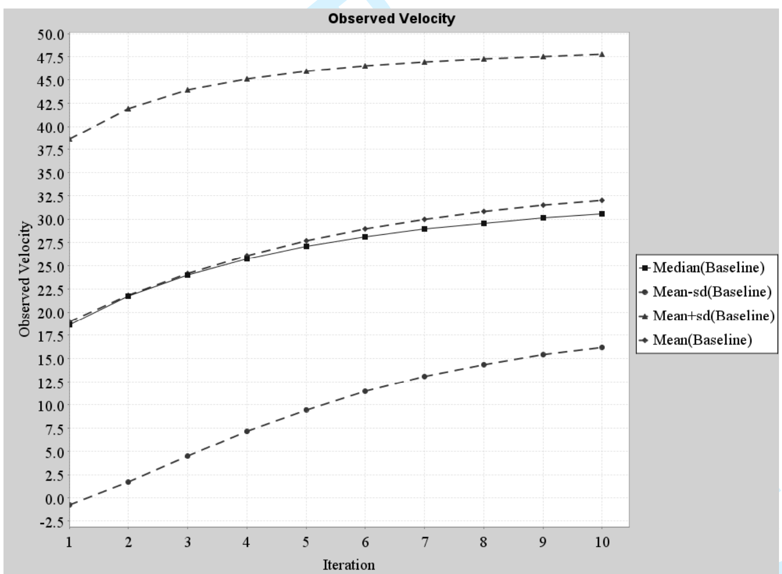

Fig. 8 Project velocity values – median, mean, mean ± 1 SD

This is our "Baseline" scenario, with no PV evidence entered into the model. By entering PV evidence, we can construct various alternative scenarios and compare the learned parameters and predicted values of future PV. The values shown in Table 2 were used to construct three such scenarios, all based on 50 hours of available effort per iteration. No values were entered for V_9 or V_{10}, allowing the model to predict these values. These represent projects that are respectively failing, performing as expected, or progressing with great success. We refer to these as the "Failing", "Average" and "Success" scenarios respectfully.

Note that the "Success" scenario uses deliberately unrealistic figures in order to test the range of the model.

Table 2 - PV values for three scenarios

Scenario\PV	V_1	V_2	V_3	V_4	V_5	V_6	V_7	V_8
Failing	2	3	3	4	4	3	4	4
Average	20	25	27	28	28	29	30	31
Successful	200	205	210	215	219	223	225	227

5.1. Parameter Learning in Different Scenarios

Fig. 9 shows the resulting distributions of the bias node, b_{10}. There are four distributions, one for each scenario. The "Failing", "Average" and "Baseline" scenarios have mean values close to one, although both the Failing and Average scenarios have reduced variances compared to the baseline. The reduced variances are to be expected from scenarios where evidence has been entered.

In Fig. 8 the Baseline scenario predicted values for V_1 to V_8 in the range 18-30. However the Success scenario entered evidence in the range 200-227, indicating that the project team has done 200-227 estimated IEDs in a single iteration with only 50 man-days of effort. Clearly this can only come about if their estimates are significantly biased, and indeed, the model suggests that the bias in this case has a mean value of 4.3. This only accounts for part of the high PV values however. The remainder is accounted for by an increased effectiveness limit (Fig. 10) which allows a greater process effectiveness.

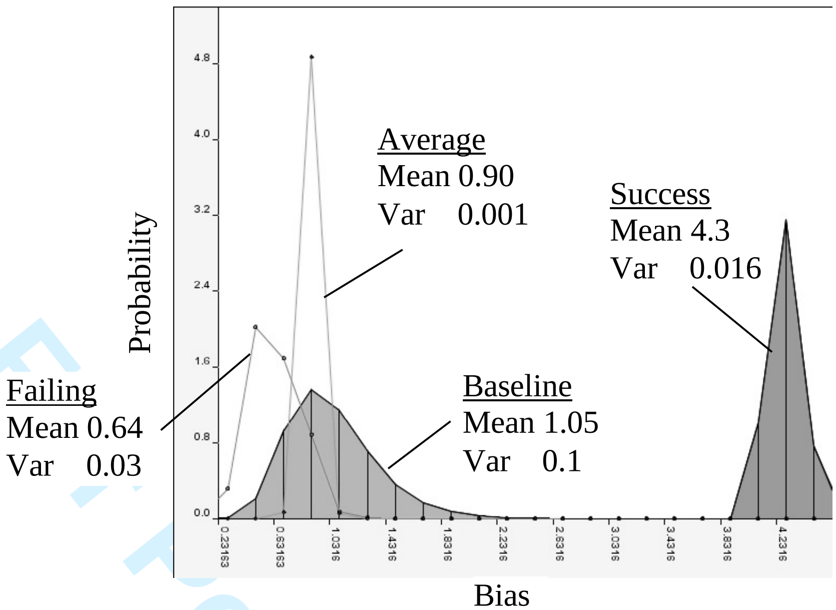

Fig. 9 Bias distribution, iteration 10

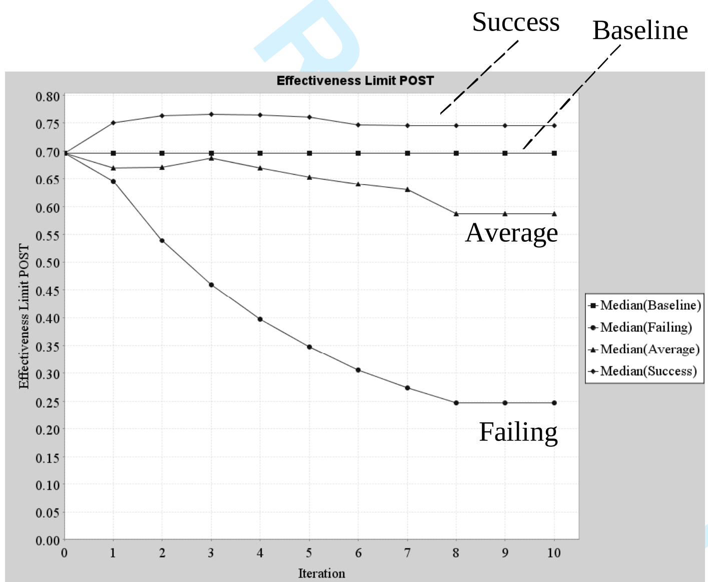

Fig. 10 Effectiveness Limit, median, 5 iterations

As we might expect, the Failing scenario shows a poor effectiveness limit and a very small improvement in process effectiveness (Fig. 11). Surprisingly, the success scenario shows an even worse process improvement. However, this is because the model is forced to assume a very high process effectiveness in the initial iterations. The values provided are so far outside the normally expected range that the model is continually trying to compensate by bringing the process effectiveness back down again. By iteration 6 the process improvement finally begins to stabilize.

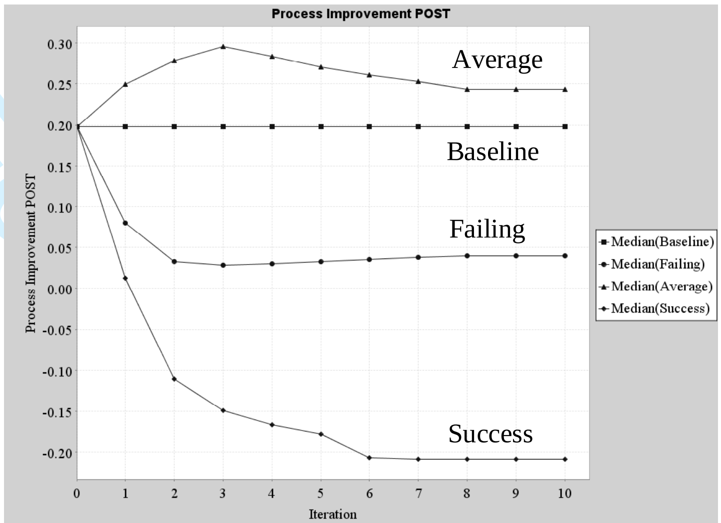

Fig. 11 Process Improvement, median, 5 iterations

Both the Effectiveness Limit (Fig. 10) and the Process Improvement (Fig. 11) change as evidence is entered in the first eight iterations. The model therefore learns as new evidence is entered and changes its predictions accordingly.

Fig. 12 shows the behavior of the Bias node, b_i, in the Average scenario. 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.

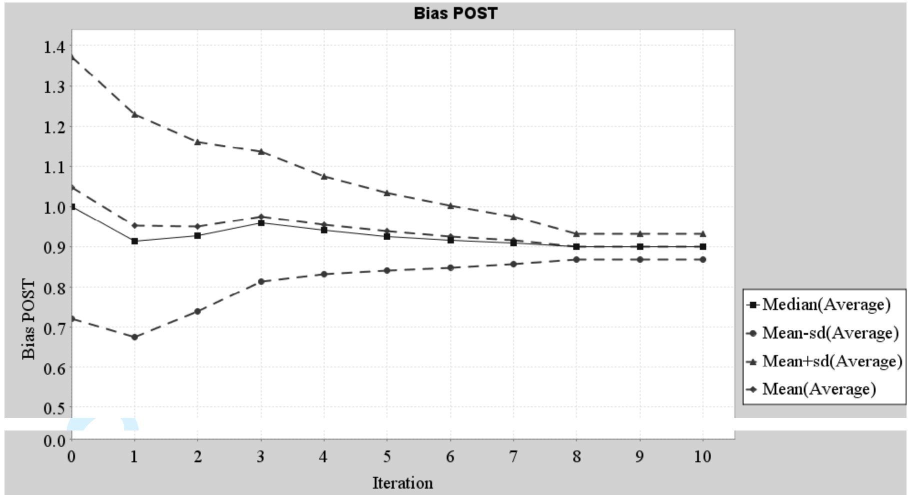

Fig. 12 Bias, Average scenario, median, mean ± 1 SD

5.2. 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.

XP practices cannot be categorized as simply being "implemented" or "not implemented". There are degrees to which various practices are adopted. For example, a team may choose to program in pairs for complex parts of the code and program individually when writing routine code. It is important therefore that XP practices are represented by nodes with a sufficient range of states to reflect the degree of variation of that practice within the project.

An indicator node for the Effectiveness Limit is shown in Fig. 13: the "Collective ownership" node. This is the extent to which collective code ownership is practiced. It is a ranked node, consisting of five discrete values ranging from Very Low to Very High. Ranked nodes allow the user to enter a range of values for "Collective Ownership". The probability of these five values is derived from a truncated normal distribution whose mean is l_i, and whose variance is arbitrarily set to 0.1. This distribution ensures that a high degree of collective ownership 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].

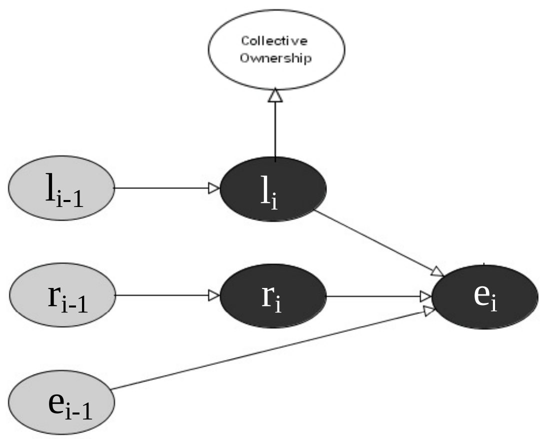

Fig. 13 The "Collective Ownership" indicator node

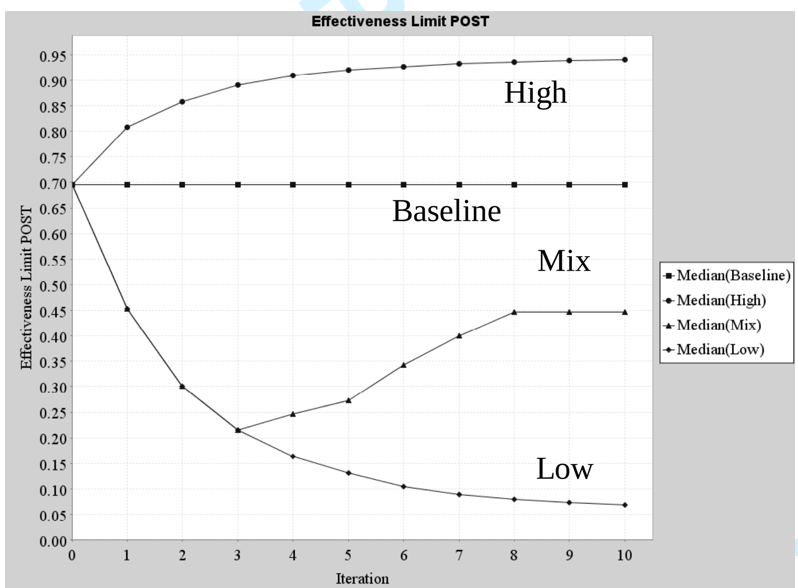

Fig. 14 Effectiveness Limit with and without indicator node evidence

With no evidence, the node plays little part in the model, and its parent, l_i, remains constant from one iteration to the next (the "Baseline" scenario). However, when we set the value of "Collective ownership" in each iteration to "Very High" (the "High" scenario) then the situation changes. The evidence back propagates to l_i. Because of the learning mechanism described above, the effect is cumulative and the mean value increases across iterations. The difference is shown in Fig. 14.

Values entered into this node are examples of expert judgment. The ease with which expert judgment can be combined with objective evidence and prior assumptions is one of the benefits of the Bayesian Network approach to modeling.

Two other scenarios are also shown, one where the Collective Ownership node is always set to "Very Low" (the "Low" scenario) and a slightly more realistic case (the "Mix" scenario). In the Mix scenario, Collective Ownership starts off "Very Low". However management realize that there is a problem and take steps to improve collective ownership. By iteration 4 Collective Ownership improves to "Medium" and by iteration 6 it achieves a "High" value.

The extent to which XP practices are implemented can therefore have a dramatic effect on the model parameters, which in turn propagates through to the model's predictions.

It is not necessary to include all XP practices as indicator nodes in all iterations. If a practice, such as pair programming say, is consistently maintained at the same level in all iterations, then its effect will be included in the learned values of the model parameters. Only practices which affect project velocity and which vary significantly between iterations, need to be included as indicator nodes.

6. Model Validation

We apply the model to an industrial case study (section 6.1). The model can learn from the initial data entered from the project (section 6.2) and adjusts its predictions once beneficial XP practices are taken into account (section 6.3). Section 6.4 provides an example of how the model can be calibrated for a specific XP practice. Finally, in Section 6.5 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.

6.1. 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 over the duration of the project. The full data set is shown in Table 3.

Table 3 – 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 3 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. 15 shows the resulting marginal distributions which are generated for the V_i node. There is one distribution for the node in each timeslice.

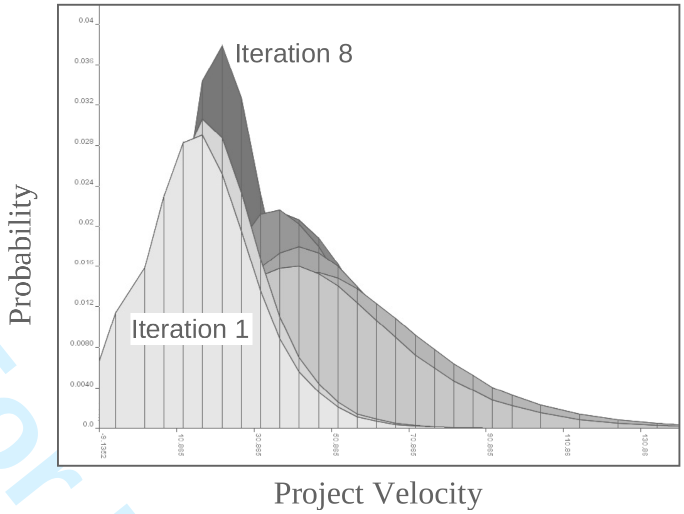

Fig. 15 Distributions for V_i , one per timeslice

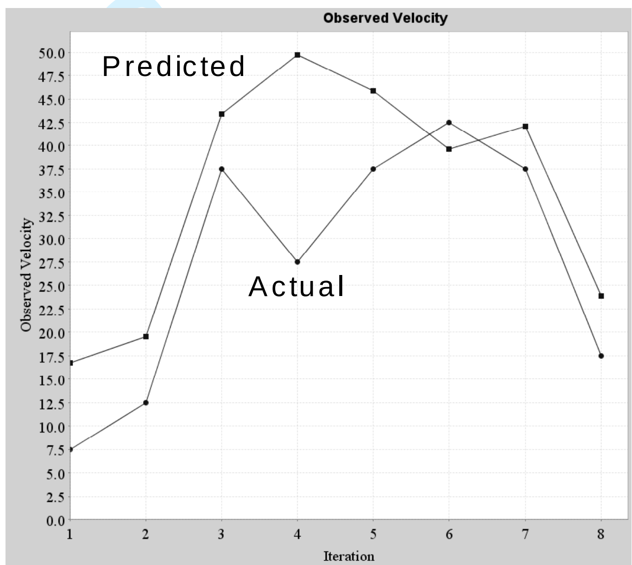

Fig. 16 Predicted vs. actual Motorola *V* (medians)

The median values from the V_i distributions are shown in Fig. 16 (the “Predicted” graph). Actual values for $\underline{V}_i$ are shown in the same figure for comparison (the “Actual” graph). The large “Actual” dip in iteration 4 is put down to a post-Christmas malaise by the Motorola team.

6.2. Parameter Learning

There are a number of problems with the predicted values in Fig. 16. 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.

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.

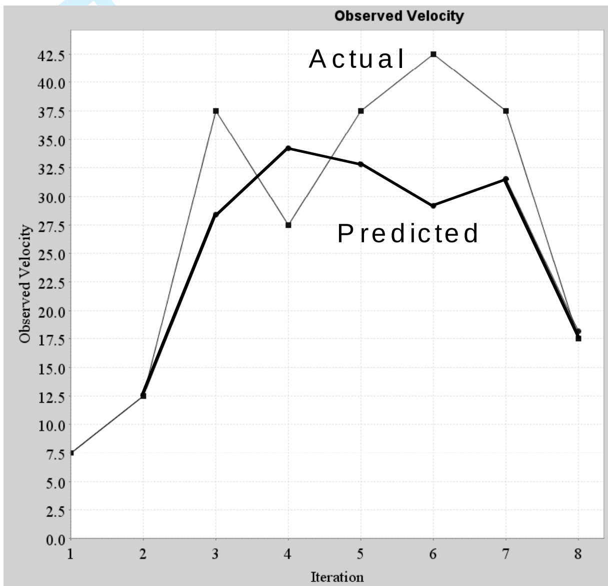

Fig. 17 Predicted vs. actual *V*, 2 observations

The graphs in Fig. 17 show the change in predicted values when $\underline{V}_1$ and $\underline{V}_2$ have been entered. The whole of the "Predicted" graph moves to lower values as the model learns from the observations. The predictions for V_3 and V_4 improve as a result. However, the predicted values for V_5, V_6 and V_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.

6.3. "Onsite customer" as an Indicator Node

An indicator node for the Effectiveness Limit is shown in Fig. 18: 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].

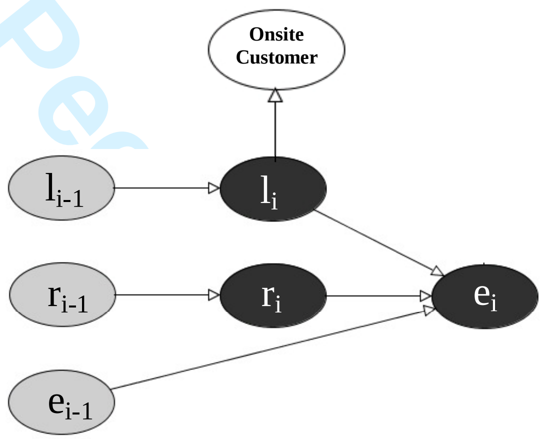

Fig. 18 The "Onsite Customer" indicator node

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.

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.

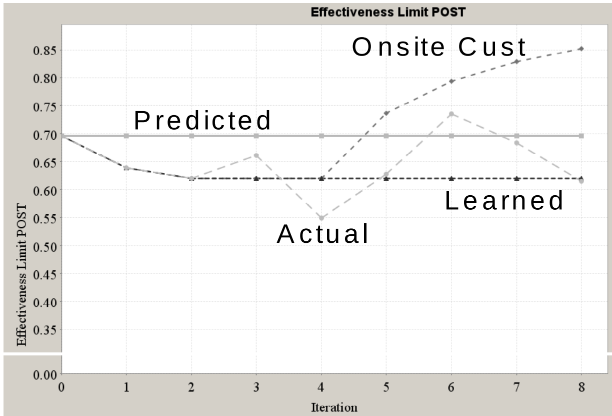

Fig. 19 Effectiveness Limit with and without indicator node evidence

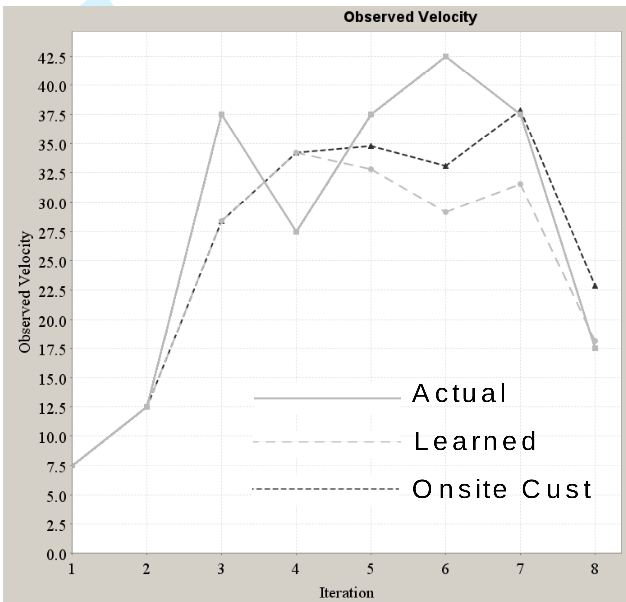

Fig. 20 *V* with and without Onsite Customer evidence

Fig. 19 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. 17.

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. 20.

6.4. 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.

Our model does not explicitly include details of defect fixing effort (including requirements defects); 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 "Miscellaneous Effort", m_i, to be the fraction of effort that does not contribute to completed user stories: $E_i = V_i + m_i$.
2. Miscellaneous effort is composed of a variable component due to defect fixing effort, d_i, and a set of fixed overheads, o_i: $m_i = d_i + o_i$. This does not provide a full description of

miscellaneous effort, but it is adequate for this model.

3. When the onsite customer input is at its maximum, the rework effort is at its minimum.

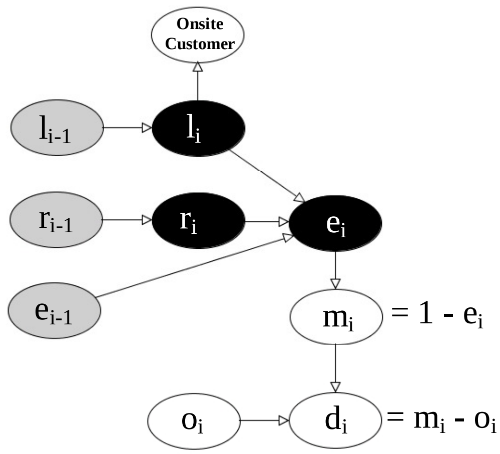

Fig. 21 BN used to calibrate the Onsite Customer node

With these assumptions in place, we can use the BN shown in Fig. 21 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 $\underline{o_i}$ 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.

The resulting defect effort percentages for each value of “Onsite Customer” across four iterations are shown in Fig. 22. These are similar to the empirical curves of Figure 3 in [9].

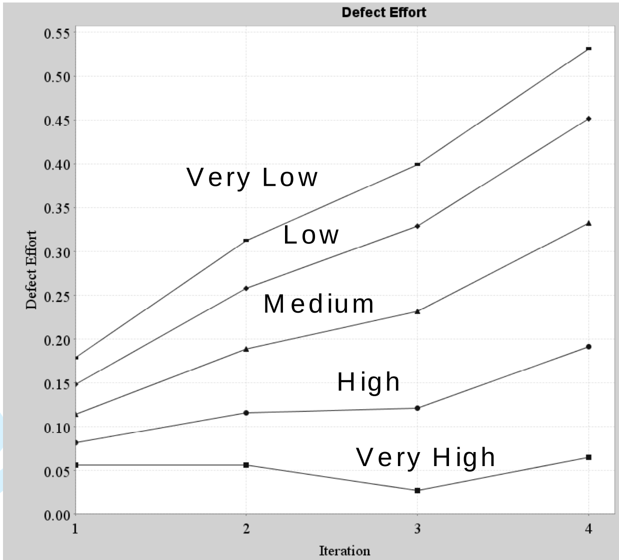

Fig. 22 Defect effort % for each Onsite Customer setting

6.5. Timescale Prediction

Fig. 23 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.

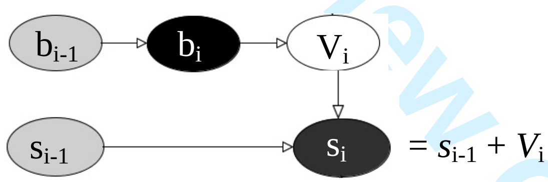

Fig. 23 Project Velocity summed to date

Plots of s_i for the initial prediction, the learned prediction and the actual scenarios are shown in Fig. 24. 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.

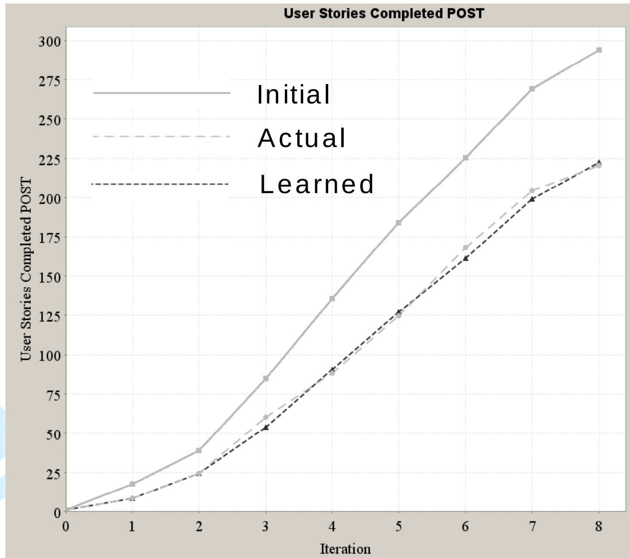

Fig. 24 Sum V_i to date

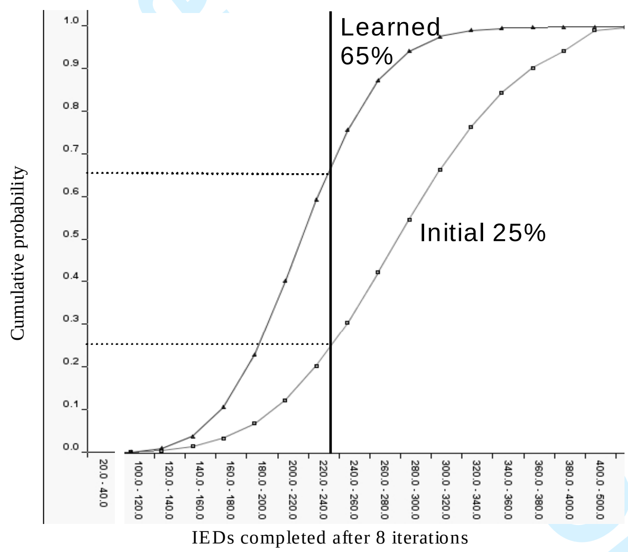

Fig. 25 Iteration 8 cumulative distributions

The Motorola project completed 224 IEDs of functionality before the project ended. The model can quantify the uncertainty involved in completing 224 IEDs within 8 iterations. Fig. 25 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 224 IEDs by the end of the 8th iteration. For the “Initial” scenario, there is only a 25% chance of completing up to 200 IEDs. Once the model has learned from $\underline{V}_1$ and $\underline{V}_2$, the probability is revised up to a 65% probability. This means that the model was initially too optimistic in its predictions (a 65% chance of delivering up to 224 IEDs means a 35% chance of delivering *more than* 224 IEDs).

7. Conclusions and Discussion

We have developed a model of XP project velocity and shown that it reproduces known empirical behavior from iterative projects.

The model 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. Other XP practices can be incorporated in the model using similar techniques.

While the model presented here has successfully demonstrated the benefits of using a learning BN model in XP projects, we recognize that there are a number of threats to its validity.

1. The model relies on having sufficient degrees of freedom to learn from its environment. This is principally accomplished by updating the parameter nodes l_i and r_i. It is possible those are insufficient to accommodate the full range of behaviors of real XP projects, or that some future XP practices cannot be wholly accommodated as indicators of one of these nodes.
2. Only a single industrial test case has been used. Greater confidence in the model will be achieved through exposure to a greater variety of data sets.
3. The example shown had the benefit of real effort data from a completed project. At the start of a project, only projections of available effort are available.

4. No sensitivity analysis has been performed on the model priors in Iteration 0. This is not an especially serious concern because, regardless of the initial values, the model will adapt to the current project's local conditions as soon as the first few iterations are completed. Clearly, any change in the means or standard deviations of the priors will affect the model's initial predictions. We would expect that more mature software development organizations would replace the supplied values with distributions based on their own previous metrics programs.
5. Two XP practices have been included in the model: "Collective ownership", using hypothetical data, and "Onsite customer", using data from a single study. Empirical data on the effectiveness of other XP practices needs to be used in order to calibrate appropriate indicator nodes.

Despite these concerns, there are a number of clear benefits to this approach.

1. Although prior metrics information is valuable, an extensive data collection phase is not essential. The model starts off making generic predictions, but quickly alters them as local data becomes available. Developers tasked with metrics collection therefore see an immediate benefit from doing so: predictions about their own project will improve as a result. Contrast this with traditional metrics collection programs, which often founder because of the need for long-term commitment.
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. Provided suitable empirical evidence is available, it is relatively simple to add new XP

practices or other environmental features, making the model extremely versatile.

The model presented here differs from many of the other causal models described in section 2.2. Rather than trying to construct a complex graph of causal relationships, it opts instead for a very simple structure. This model recognizes that, for a large variety of reasons, software productivity varies throughout the iterations of an agile project. It therefore learns the cumulative effect of these variations rather than trying to model their interactions explicitly.

Users of the model only need to provide three items of information:

1. available effort over the timescale of the project,
2. measured project velocity as it becomes available,
3. the extent to which XP practices are varying between iterations.

The first two should be available anyway in any XP project and the third can be supplied using subjective judgment. The burden to developers and managers in maintaining this model is therefore minimal. In return for this small overhead, projects get improved PV predictions such as in Fig. 24 and a quantitative assessment of the risk, as in Fig. 25.

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.

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] N. E. Fenton, and M. Neil, "A Critique of Software Defect Prediction Models," IEEE Transactions on Software Engineering, 25(4):675-689, September 1999
[11] Fenton NE, Neil M and Caballero JG, "Using Ranked nodes to model qualitative judgements in Bayesian Networks" to appear in IEEE TKDE 2007, 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
[13] 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
[14] 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
[15] Neil, M. and Fenton P. Improved Software Defect Prediction. 10th European SEPG, London, 2005.
[16] S.Bibi, I.Stamelos, Software Process modeling with Bayesian belief Networks, 10th International Software Metrics Symposium Chicago, September 2004
[17] Wooff D.A., Goldstein M., Coolen F.P.A., Bayesian Graphical Models for Software Testing, IEEE Transactions on Software Engineering, Vol 28, Issue 5, pp. 510-525
[18] Williams, L. and Erdogmus, H., On the Economic Feasibility of Pair Programming, International Workshop on Economics-Driven Software Engineering in conjunction with the International Conference on Software Engineering, May 2002.
[19] Ross, S. A. Fundamentals of Corporate Finance, Irwin/McGraw-Hill, 1996.
[20] Beck K, Andres A, Extreme Programming Explained Embrace Change, Addison-Wesley Professional; 2 edition (November 16, 2004)
[21] Beck K, Extreme Programming Explained, Embrace Change, Addison-Wesley Professional; 1st edition (2000)
[22] Extreme Programming Installed, Ron Jeffries, Ann Anderson, Chet Hendrickson, Addison-Wesley Professional ; 1st edition.
[23] A. Cockburn and L. Williams. The costs and benefits of pair programming. In eXtreme Programming and Flexible Processes in Software Engineering XP2000, Cagliari, Italy, June 2000.
[24] J. Nosek. The case for collaborative programming. Communications of the ACM, 41(3):105–108, Mar. 1998.
[25] L. Williams, R. Kessler, W. Cunningham, and R. Jeffries. Strengthening the case for pair-programming. IEEE Software, pages 19–25, July/Aug. 2000.
[26] Padberg F, Müller M, Analyzing the Cost and Benefit of Pair Programming, Proceedings of the Ninth International Software Metrics Symposium (METRICS'03)
[27] Misic, V., Gevaert, H., Rennie M. (2002) "Extreme dynamics: modelling the extreme programming software development process ". Workshop on empirical evaluation of agile processes, XP/Agile Universe 2002
[28] Abdel-Hamid T, "The Dynamics of Software Projects Staffing: A System Dynamics Based Simulation Approach," IEEE Transactions on Software Engineering, vol. 15, no. 2, pp. 109-119, 1989
[29] Kuppuswami, S., Vivekanandan K., and Paul Rodrigues (2003): A System Dynamics Simulation Model to Find the Effects of XP on Cost of Change Curve. In proceedings of Fourth International Conference on Extreme Programming and Agile process in Software Engineering, (XP2003), May 25--29, 2003, Genova, Italy.
[30] Lepar V, Shenoy PP, A Comparison of Lauritzen-Spiegelhalter, Hugin, and Shenoy-Shafer Architectures for Computing Marginals of Probability Distributions (1998), Proceedings of the 14th Conference on Uncertainty in Artificial Intelligence (UAI-98)
[31] Agile Manifesto, http://www.agilemanifesto.org/. Accessed 10 May 2007.
[32] Alessandra Cau, Giulio Concas, Marco Melis, Ivana Turnu, Evaluate XP Effectiveness Using Simulation Modeling, Proceedings 6th International Conference Extreme Programming and Agile Processes in Software Engineering, XP 2005, Sheffield, UK, June 18-23, 2005.
[33] B. George and L. Williams, "An Initial Investigation of Test-Driven Development in Industry", Proceedings of ACM Symposium on Applied Computing, Melbourne, FL, pp. 1135-1139, 2003.
[34] Hao Wang, Fei Peng, Chao Zhang, Andrej Pietschker, Software Project Level Estimation Model Framework based on Bayesian Belief Networks, Sixth International Conference on Quality Software (QSIC'06)
[35] Stamelos, L. Angelis, P. Dimou, and E. Sakellaris. On the use of bayesian belief networks for the prediction of software productivity. Information & Software Technology, 45(1):51–60, 2003.
[36] Albrecht A.J., "Measuring Application Development Productivity," Proc. Joint SHARE/GUIDE/IBM Application Development Symp.,pp. 83-92, 1979

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 a professor of computing at Queen Mary, University of London, and CEO of Agena, which specializes in risk management for critical systems. His research interests include software metrics, formal methods, empirical software engineering, software standards, and safety-critical systems: recent projects focused on using Bayesian belief nets and multicriteria decision aid for risk assessment. He has a BSc from the University of London and an MSc and PhD from Sheffield University, all in mathematics. Contact him at Queen Mary, Univ. of London, Mile End Rd., London E1 4NS, UK; norman@agena.co.uk.

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.