Improved prediction in Extreme Programming Projects using Bayesian Networks

Peter Hearty, Norman Fenton, David Marquez, Martin Neil

***Abstract*--Causal models (Bayesian networks) have been used with some success to provide software managers with improved risk assessment and quality assurance methods. It is possible to provide more intuitive and accurate predictions of key project attributes such as effort and defects because they take account of causal (process) factors. To date these methods have largely been restricted to projects using traditional development approaches such as waterfall or spiral. Yet agile software development methods, such as Extreme Programming, are becoming increasingly popular alternatives and have just as great a need for accurate predictions and risk assessment. We present a novel Bayesian Net model that can provide improved effort predictions and risk assessment in an Extreme Programming environment. The model successfully reproduces real world characteristics of XP Project Velocity, is capable of considerable extension, and uses data to learn key parameters.**

***Index Terms*— extreme programming, Bayesian nets, causal models.**

1. INTRODUCTION

In [12], Fenton and Neil explain 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, and can include expert judgment in the models. They allow causal relationships to be expressed in a way that is both quantitative and intuitive. In software engineering, where data is often sparse or of uncertain quality, BNs provide the ideal tool for expressing the complex relationships between the software development process and the products that result.

Fenton, Neil, and others have gone on to develop a series of BN models, culminating in the AID tool [15], the MODIST models [16], and the extensive trials of revised models at Philips

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.

[17]. 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 [13] have developed BNs modeling the software test process while Bibi and Stamelos [5] 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 Extreme Programming (XP) [1], 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 and computationally infeasible. What is needed is an extremely small core model that can be extended as needed.

In XP development, the emphasis is on producing working code through iteration and adaptation. Data collection and subsequent metric information are likely to be extremely limited and varied. Any model of an XP development environment must recognize this and not be tied to a specific set of practices.

One metric that is always available in XP, however, is Project Velocity (PV). PV is a measure of team productivity and is central to any XP project. Roughly speaking it can be thought of as "productive effort per iteration". The exact definition of PV is given in section 2.2.

The iterative nature of XP means that PV data is collected early on in the project. A key requirement of XP is fast feedback, therefore we aim to incorporate this data into the model, enabling it to learn key parameters and increase the confidence of predictions for subsequent iterations.

We therefore aim to produce a model that satisfies the following requirements.

1. It must be able to monitor and predict PV, taking into account the impact of relevant process factors.
2. 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. It must be possible to adapt the model to handle different types of data for different environments. In particular, the model should be capable of including all the key practices of XP.
4. Many projects report low initial productivity, gradually rising on subsequent iterations [18], [19] and [10]. The model should be capable of replicating this empirical behavior.
5. The model must learn, 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 model discussed here has been implemented using the AgenaRisk toolset [7]. 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 gives a brief overview of the main ideas in XP that are relevant to this model. Section 3 gives some background on Bayesian Nets. Section 4 describes the main iterative part of the model and the model used to set the initial conditions. Section 5 gives some results of the model's behavior. Finally, Section 6 provides some conclusions and indications of future work.

2. EXTREME PROGRAMMING

Extreme Programming (XP) is an approach to software development that consists of a collection of values, principles and practices as outlined by Kent Beck, Ward Cunningham and Ron Jeffries [1][2]. These include most notably: 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. Instead, the customer and development team agree a System Metaphor that summarizes the aims of the project, and a series of User Stories (described in Section 2.1) which concisely define the requirements.

XP is a highly iterative approach. Beck recommends iterations of a fixed, one week duration [1] (p.46). Each iteration should deliver a usable version of the system, updated to include the user stories allocated to that iteration. The effort required to deliver the user stories is what characterizes the PV (described in Section 2.2). The notion of misdirected effort is described in Section 2.3, while in Section 2.4 we define some key process factors that are used in the subsequent models.

2.1. User Stories

User stories are the main unit of work in XP. They are written by the customer who assigns a priority to each user story. The allocation of user stories to an iteration is done by the customer on the basis of the user stories' priorities. The customer also defines the acceptance tests that a user story must pass. This provides a clear definition of what constitutes completed work.

Developers assign the effort that they believe is required for them to design, code and test the user story. Since these are personalized estimates, they already include allowances for the productivity of individual team members.

Efforts are estimated using a unit called *Ideal Engineering Days* (IEDs). An IED 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.

Example: Suppose that there are 8 user stories in total for a project prioritized as shown in Table 1. Suppose that in the first iteration the developers feel they will have a total of 5 IEDs available. Then they will plan to complete the first three user stories *US1*, *US2* and *US3* in iteration one.

Table 1 Estimating user stories

User Story	Priority	Ideal Engineering Days	Iteration
US1	1	$U_1^1 = 2$	1
US2	2	$U_1^2 = 1$	
US3	3	$U_1^3 = 2$	
US4	4	$U_2^1 = 3$	2
US5	5	$U_2^2 = 2$	
US6	6	$U_3^1 = 4$	3
US7	7	$U_3^2 = 1$	
US8	8	$U_4^1 = 1$	4

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.

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

It is important to emphasize this is an *estimate* of how many IEDs the team thought it would take to complete the work. It is not the number of actual person-days taken, nor is it the calendar time taken. Thus, in the previous example, if the first three user stories really were completed in iteration 1 then $V_1 = 5$.

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. Expressed more concisely: we can expect to be as productive in the next iteration as we were in the last. So, using the example shown in Table 1 we have $V_1 = 5$, and so we would schedule tasks US4 and US5 for iteration 2 since the estimated IEDs required for these two user stories is 5. If it turns out that in schedule 2 we only manage to complete US4 then $V_2 = 3$.

Note that the actual time taken to complete a user story is not used here. To illustrate why it is unnecessary, let us suppose that the developers working on *US*1, *US*2 and *US*3 had carefully filled in time sheets and determined that the time spent on those stories was not the 5 days that was estimated, but actually took 6 IEDs, i.e. there was a bias, b_1, in their estimates. (Note that the use of the word "bias" here is not intended in the statistical sense of a biased estimator).

If A_1^j are the actual efforts taken then:

$$b_1 = \frac{\sum_j U_1^j}{\sum_j A_1^j} = \frac{V_1}{\sum_j A_1^j} \qquad \textbf{Eq. 2}$$

In this case, b_1 is 5/6. A project manager might assume that the remaining tasks have been underestimated by the same amount and multiply them by 6/5 to compensate. *US*4 and *US*5 would then have new estimates of 3.6 and 2.4 respectively. The manager knows that the team did 6 actual IEDs work in iteration 1 and, all things remaining equal, are likely to do 6 actual IEDs in iteration 2, so he schedules tasks whose updated estimates add to 6. This would result in US4 and *US*5 being scheduled for the next release - exactly the same as we had before (except that there is

now a lot more time tracking and form filling being done).

This scheduling mechanism assumes that the ratio of effort (people × working days) to PV 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. If there is no consistent bias in the effort estimations, i.e. there is as much overestimation as underestimation, then 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.

If the next set of user story estimates do not sum to exactly V_i then various options are possible.

- One of the user stories can be broken down further. In fact, this happens anyway in XP. Each user story is broken down into development tasks as a result of a short design stage. This level of granularity is not included in this model.
- An alternative user story can be selected, although this breaks the prioritization guidelines in XP.
- Stories with estimates slightly less than V_i can be chosen.

It is recognized that in real world projects there are dependencies between iterations. Efficient implementation sometimes demands that the order of work differ from the customer's desired prioritization. There are also rework tasks which arise when work in a previous iteration must be revisited. None of this is explicitly included in this model. Inter-iteration dependencies are

simply subsumed within other, more general, overheads.

The introduction of pair programming has no effect on how these calculations are performed. If the developers in our example are paired then the number of user stories completed in iteration one might decrease, say to a set of user stories whose estimates totaled 3 IEDs instead of 5. User stories adding up to 3 IEDs are then scheduled for the next iteration. Similarly, the extent to which user stories are complete serially or in parallel has no effect on the model.

2.3. Misdirected Effort

To help express the relationship between successive values of *V*, we introduce the concept of Misdirected Effort, *M*. Misdirected effort is effort which does not result in completed user stories. This includes both development and non-development activity (such as administrative tasks).

All user stories scheduled for inclusion in an iteration, which fail their acceptance tests, form part of the misdirected effort. This can happen for several reasons.

1. Software defects which are missed during unit testing may be serious enough to cause the acceptance tests to fail. This can be caused by: poor development, poor unit testing, good acceptance testing.
2. The developers can have misunderstood the requirements. Poor quality and insufficient quantity of customer input and feedback might increase the probability of this occurring.
3. Underestimation of the effort required to complete a user story can mean that development tasks are not completed on schedule.

Misdirected effort is expected to be larger at the start of an XP project. Examples of this can be seen in, [18], [19] and [10]. Some of the reasons for this are unique to XP, others are common to

all software development.

- Developers may only recently have been formed into a team and may need time to learn each others' capabilities and strengths.
- The development environment may not be fully defined or there may be overheads associated with developers learning to make the best use of the tools available.
- Domain knowledge may be sparse. This might be particularly true in an XP environment where there is no comprehensive written requirements to read.
- Developers new to XP may take time to become comfortable with some XP practices such as pair programming.

This latter effect has been reported, for example, by Williams, Kessler, Cunningham, and Jeffries [20], and Cockburn and Williams [21], where an initial 60% drop in programmer productivity quickly recovered to only a 15% drop. This adjustment period is referred to as *Pair Jelling*. This is consistent with work reported by Nosek [9] which showed a 50% drop in productivity but only over short, one-off assignments.

Strictly speaking, misdirected effort consists of three distinct types of component.

1. Scheduled development tasks which were not completed. These must be carried over into the next iteration.
2. Fixed administrative overheads.
3. Overheads which scale with the length of an iteration.

Two of these three components are not dependent on the length of an iteration, one is. In order to simplify the model, we restrict ourselves to fixed length iterations.

2.4. Process factors

The preceding analysis splits available effort into PV and Misdirected Effort. Some way is needed to control the amount of effort allocated to these two values. In this model, there is a single controlling factor which we call Process Effectiveness, *e*. This is a real number in the range [0,1] which represents the proportion of the available effort that translates into the PV. A Process Effectiveness of one means that all available effort becomes part of the PV and there is zero Misdirected 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. The name arises from the fact that we expect the Process Effectiveness to generally improve from one iteration to the next, or at least, not get any worse. However, to allow for failing projects, the Process Improvement can also take on negative values.

The Effectiveness Limit recognizes the fact that there are often limits to how productive a team of people can be. There will always be some necessary overheads which prevent some of the available effort being translated into PV. 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.

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).

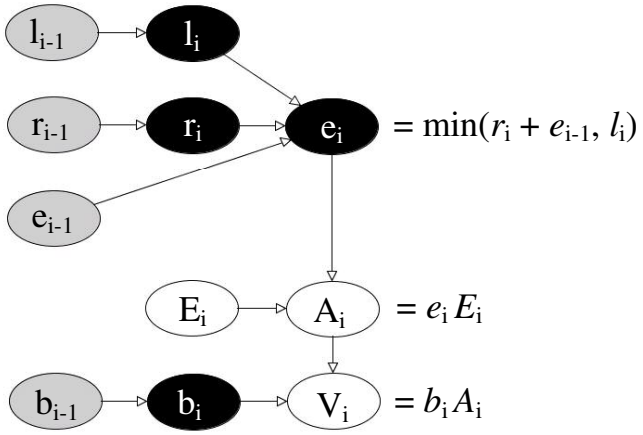

Fig. 1. Project velocity model.

Table 2 summarizes the model variables. 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 PV in iteration 2. Where the iteration is not important, we drop the subscript and refer to V, for example, as the PV.

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 [4] and [11].

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 PV at iteration three, then $\underline{V}_3$ is the measured value.

Table 2 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. $e_i \in [1, \infty)$.
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. $b_i = V_i / A_i$, $b_i \in [0,\infty)$.
M_i	Misdirected effort in iteration *i*. This is effort which did not result in completed user stories. $E_i = V_i + M_i$, $M_i \in [0,\infty)$.
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 2 are shown in Fig. 1. Several of the variables are included only to make the definitions of others more rigorous (*d*, *p* and *M*). 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 [22]. 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 conditional probabilities of nodes in $\mathbf{Z}_t$ depend only on the distributions in $\mathbf{Z}_t$ and $\mathbf{Z}_{t-1}$. 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 [11].

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).

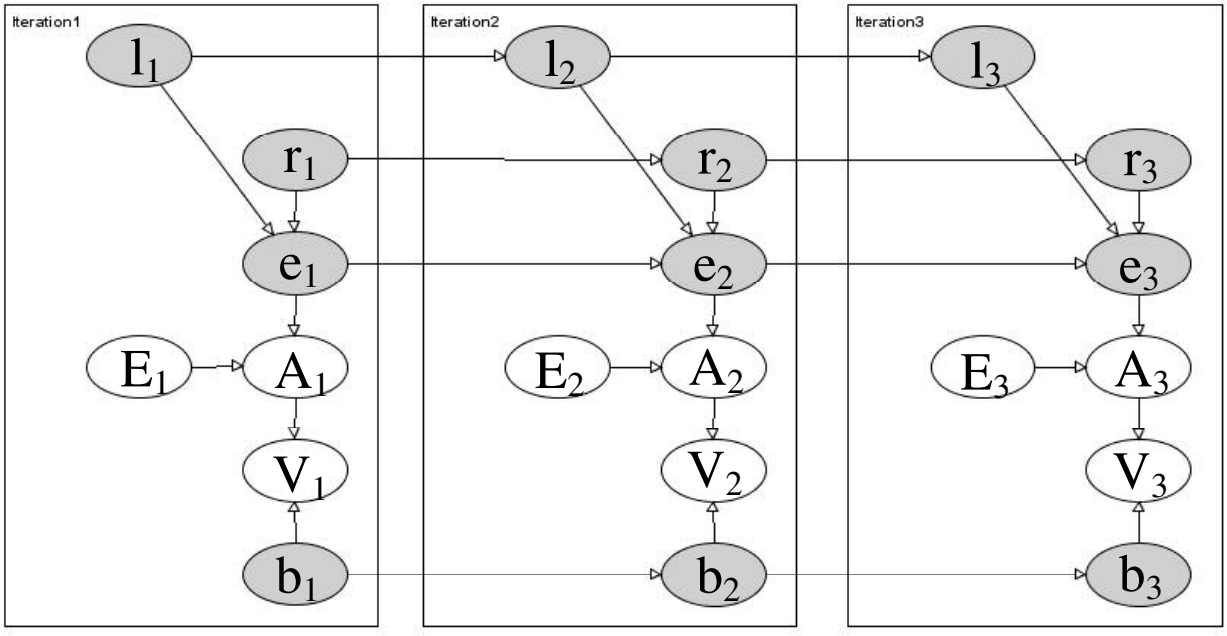

Fig. 2 Model as a DBN

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 misdirected effort and the PV. 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 PV.

4. MODEL DESCRIPTION

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

4.1. Iteration Model

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

Fragment 1 controls the Productive Effort. 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 PV. Process Effectiveness increases or decreases based on the value of the Process Improvement (r_i). It is constrained to the range [0, l_i].

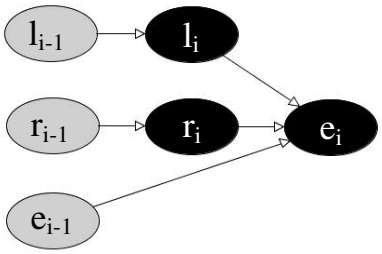

Fig. 3 Fragment 1 - Process effectiveness nodes

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}.

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). Misdirected Effort (M_i) is not shown, but it is simply the difference between E_i and 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.

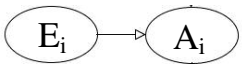

Fig. 4 Fragment 2 - Effort nodes

Fragment 3 holds the project velocity (Fig. 5). PV 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 their was no bias then the productive effort A would be the same as the PV 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.

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 [19], 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.

Evidence is entered in all of the E_i nodes.

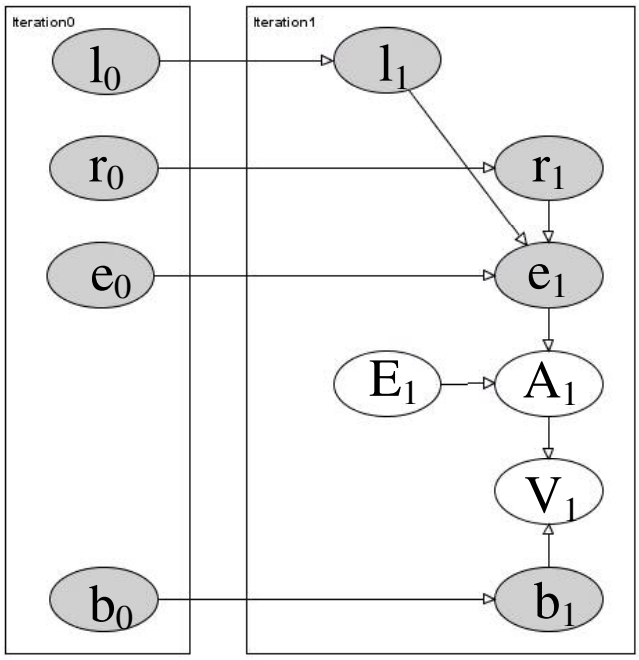

Fig. 6. Initial Velocity model

5. MODEL BEHAVIOR

Fig. 7 shows the predicted values of the PV for a project with 50 hours of effort available in each iteration. 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.

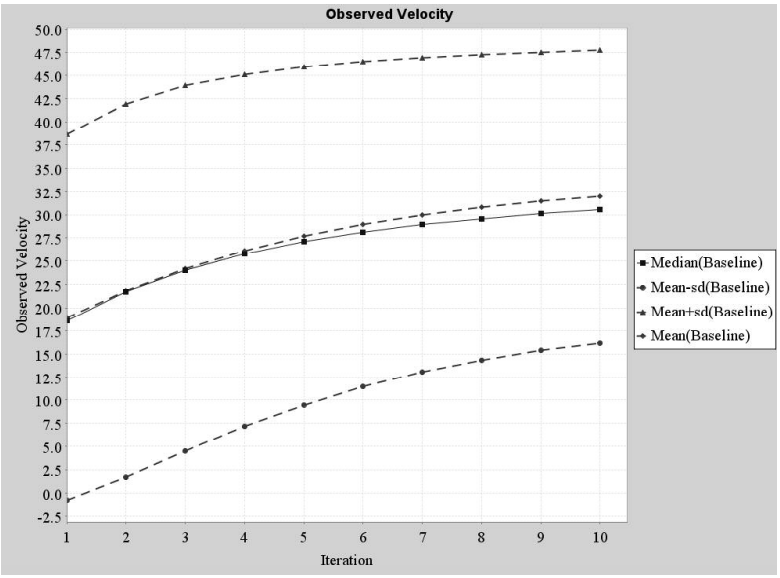

Fig. 7. Project velocity values – median, mean, mean ± 1 SD

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 3 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.

Table 3 - 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. 8 shows the resulting distributions of the bias node, b_{10}. There are four distributions, one for each scenario. Three of the scenarios have mean values close to one, although both the Failing and Average scenarios have reduced variances compared to the baseline. This is expected from scenarios where evidence has been entered.

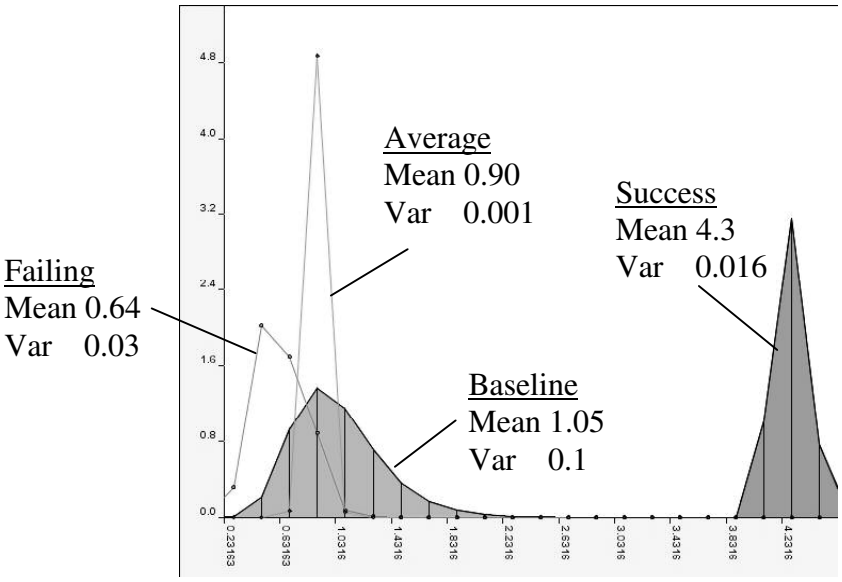

Fig. 8 - Bias distribution, iteration 10

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. These values indicate 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. 9) which allows a greater process effectiveness.

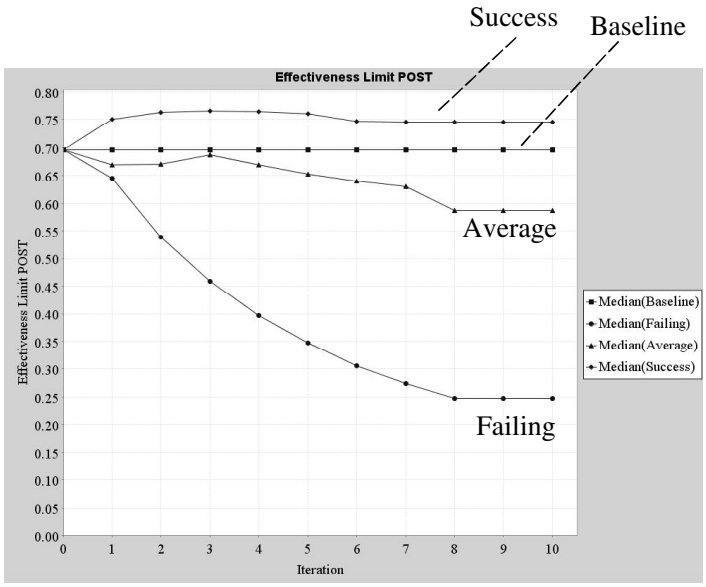

Fig. 9 – Effectiveness Limit, median, 5 iterations

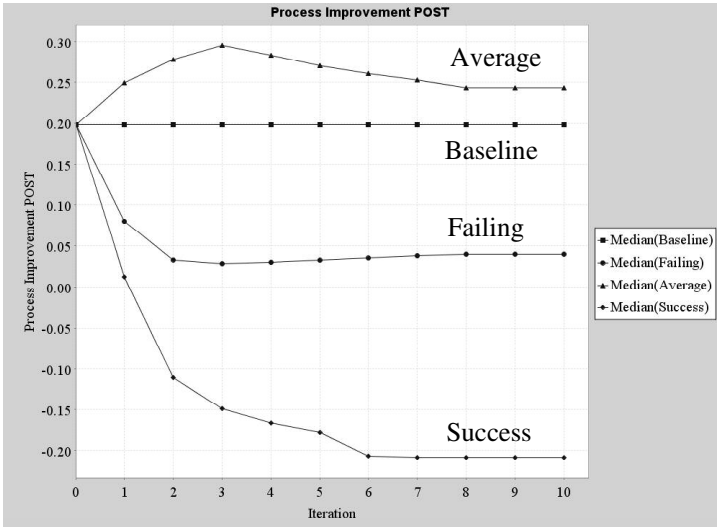

Fig. 10 - Process Improvement, median, 5 iterations

As we might expect, the Failing scenario shows a poor effectiveness limit and a very small improvement in process effectiveness (Fig. 10). 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.

Both the Effectiveness Limit (Fig. 9) and the Process Improvement (Fig. 10) 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. 11 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.

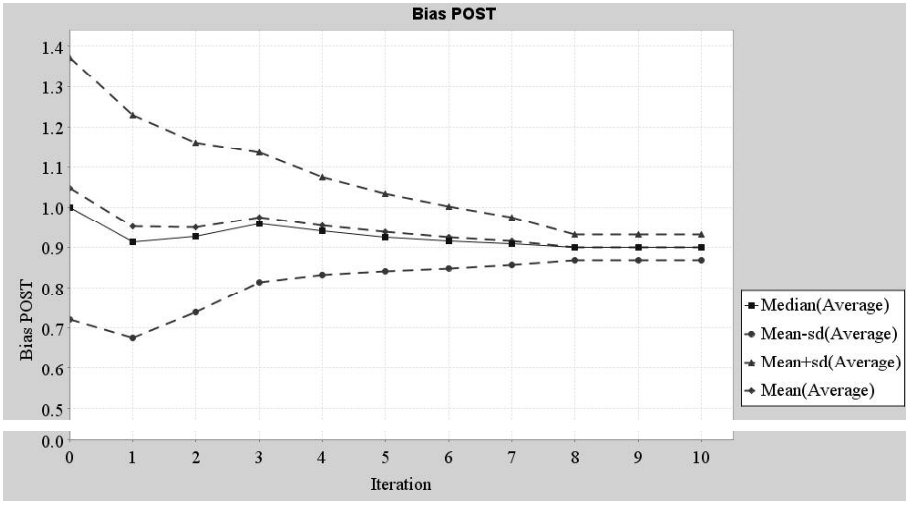

Fig. 11. Bias, Average scenario, median, mean ± 1 SD

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.

An indicator node for the Effectiveness Limit is shown in Fig. 12: 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. 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.

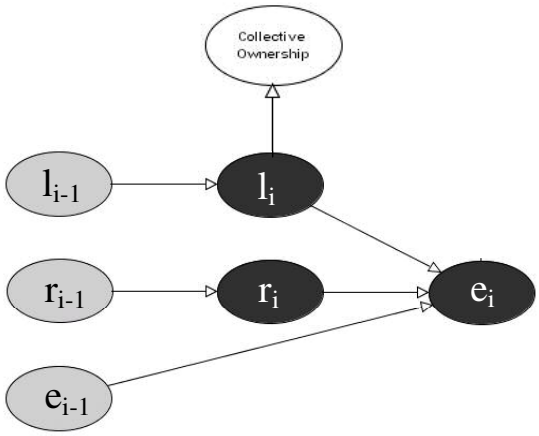

Fig. 12 The "Collective Ownership" indicator node

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. 13.

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.

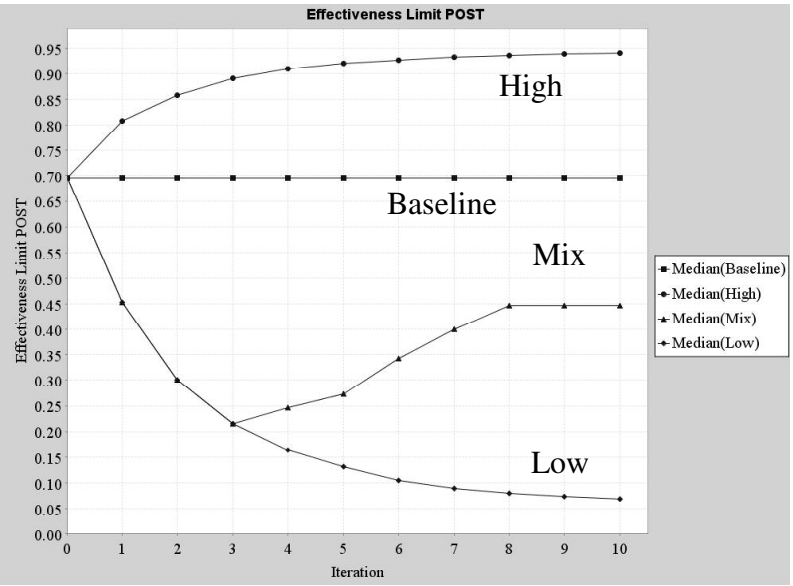

Fig. 13. Effectiveness Limit with and without indicator node evidence

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. Fig. 14 shows a slightly modified version of the PV fragment of the model. This includes an additional link node, s_i, which acts as the cumulative sum of the PV to date.

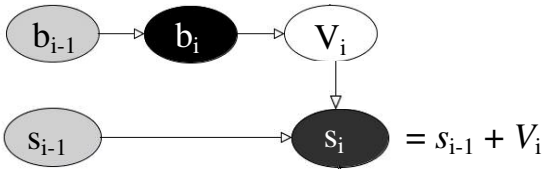

Fig. 14. Project Velocity summed to date.

Plots of s_i for the four scenarios are shown in Fig. 15. 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 for the four scenarios. The High scenario will complete in just over 6 iterations and the Baseline in just over 7. The Mix and Low scenarios will not complete within 10 iterations.

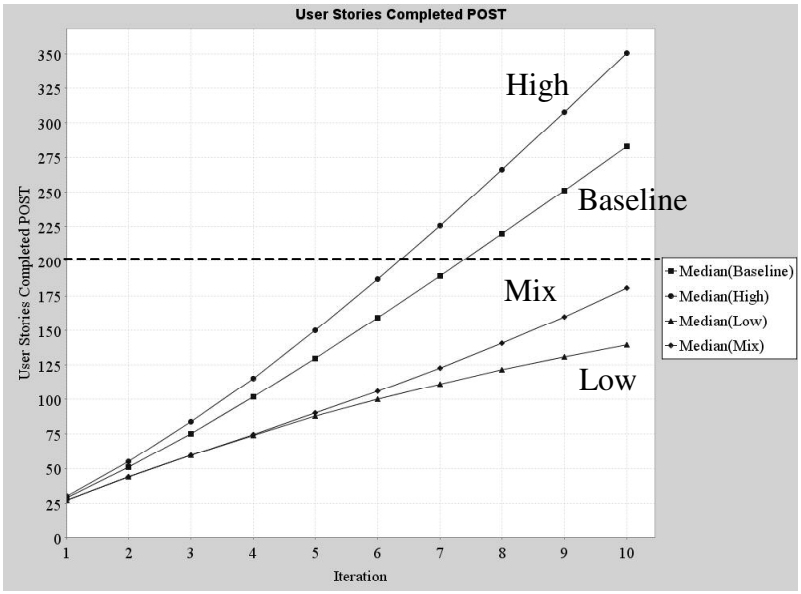

Fig. 15 Sum PV to date

The model can also quantify the uncertainty involved in completing 200 IEDs of user stories within 10 iterations. Fig. 16 shows the cumulative distribution functions for the s_i node in iteration 10. The vertical line allows us to read off the probability of completing up to 200 IEDs by the end of the tenth iteration. For the "High" scenario, there is only a 1% chance of

completing less than 200 IEDs, i.e. there is a 99% chance of completing 200 IEDs or more.

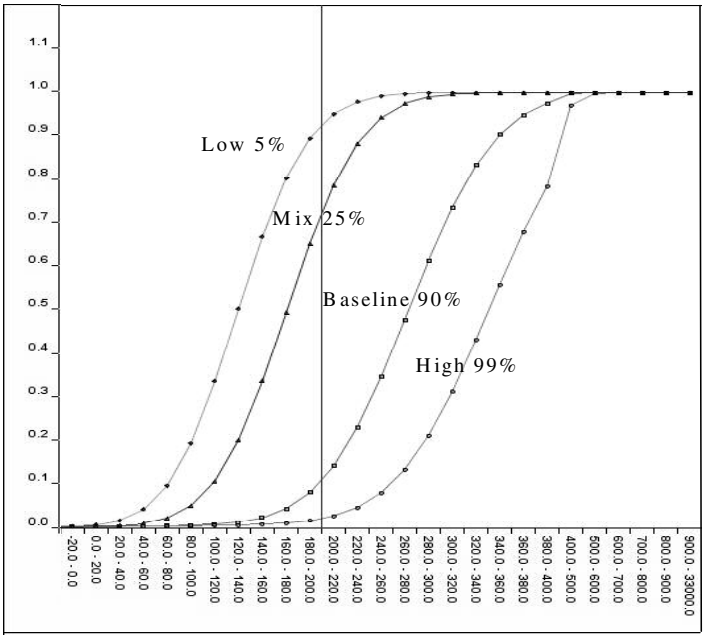

Fig. 16 Iteration 10 cumulative distributions, Sum PV to date

We have shown how the XP practice of Collective Ownership can be used as an indicator of the Effectiveness Limit. There will of course be many more factors that can be used as indicators of this parameter. Examples include the team's experience (both management and development) and the resources they have at their disposal. These, together with other XP practices, are obvious candidates for additional indicator nodes.

Similarly, other XP practices and environmental factors can be used as indicators of the process improvement, r_i. For example, the extent to which misperceptions about requirements are speedily corrected will depend on the strength of the relationship with the customer. An authoritative on-site customer voice should therefore be a strong indicator of a high r_i.

However, some XP practices cannot be included in the model at this stage. The quality of unit testing, for example, will need to be incorporated in future defect prediction models.

6. Conclusions and Future Work

We have shown that it is possible to produce a relatively simple BN model that correctly reproduces known empirical behavior in XP Project Velocity. This model is able to learn from both observations of the PV itself as well as from expert judgment entered via indicator nodes. The model is flexible in that a large number of indicator nodes and causal influences can be appended to the core model. These enable XP practices to be easily included.

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[10]. This is an important aspect of the "Brooks' factor", i.e. the tendency for larger teams to be less productive [6].

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.

References

[1] Extreme Programming Explained Embrace Change, Kent Beck, Cynthia Andres, Addison-Wesley Professional; 2 edition (November 16, 2004)
[2] Extreme Programming Installed, Ron Jeffries, Ann Anderson, Chet Hendrickson, Addison-Wesley Professional ; 1st edition.
[3] Beck K, Fowler M, Planning Extreme Programming, Addison-Wesley, 2001
[4] Jensen, Bayesian Networks and Decision Graphs, Springer-Verlag, New York, 2001.
[5] S.Bibi, I.Stamelos, Software Process modeling with Bayesian belief Networks, 10th International Software Metrics Symposium Chicago, September 2004.
[6] Brooks FP, The Mythical Man-Month: essays on software engineering, 2nd edition, Addison Wesley, 1995.
[7] AgenaRisk User Manual, Agena Limited, www.agenarisk.com.
[8] Neil M, Tailor M, Marquez D, Inference in Hybrid Bayesian Networks using dynamic discretisation (awaiting publication).
[9] Nosek JT, The case for collaborative programming, Communications of the ACM, Volume 41, Issue 3 (March 1998) Pages: 105 - 108
[10] 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)
[11] 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

[12] N. E. Fenton, and M. Neil, "A Critique of Software Defect Prediction Models," IEEE Transactions on Software Engineering, 25(4):675-689, September 1999.
[13] 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
[14] Kozlov A.V., and Koller D. 1997. Nonuniform dynamic discretization in hybrid networks, in D. Geiger and P.P. Shenoy (eds.), Uncertainty in Artificial Intelligence, 13: 314–325.
[15] 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
[16] 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
[17] Neil, M. and Fenton P. Improved Software Defect Prediction. 10th European SEPG, London, 2005.
[18] 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
[19] 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
[20] Williams L, Kessler RR, Cunningham W, Jeffries R, Strengthening the Case for Pair Programming, IEEE Software, July/August 2000 (Vol. 17, No. 4) pp. 19-25
[21] Cockburn A, Williams L, The Costs and Benefits of Pair Programming, Proceedings of the First International Conference on Extreme Programming and Flexible Processes in Software Engineering, XP2000, June 2000 Cagliari, Sardinia, Italy
[22] K. P. Murphy. Dynamic Bayesian Networks: Representation, Inference and Learning. PhD thesis, UC Berkeley, 2002.

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.