A Probabilistic Model for Predicting Software Development Effort

Parag C. Pendharkar, Girish H. Subramanian, and James A. Rodger

Abstract—Recently, Bayesian probabilistic models have been used for predicting software development effort. One of the reasons for the interest in the use of Bayesian probabilistic models, when compared to traditional point forecast estimation models, is that Bayesian models provide tools for risk estimation and allow decision-makers to combine historical data with subjective expert estimates. In this paper, we use a Bayesian network model and illustrate how a belief updating procedure can be used to incorporate decision-making risks. We develop a causal model from the literature and, using a data set of 33 real-world software projects, we illustrate how decision-making risks can be incorporated in the Bayesian networks. We compare the predictive performance of the Bayesian model with popular nonparametric neural-network and regression tree forecasting models and show that the Bayesian model is a competitive model for forecasting software development effort.

Index Terms—Bayesian belief networks, software effort estimation, probability theory.

1 Introduction

Containing the cost of software development is a major concern faced by information systems departments [25]. One of the approaches to contain cost is to develop better software effort estimation techniques. There are several software effort estimation techniques reported in the literature [35], [31]. Among a few popular techniques are linear regression models, cost models (COCOMO/COCOMO II, SLIM, etc.), neural network models, and vector prediction models [18], [31].

Chulani et al. [9] and Fenton et al. [14], [15], [16] have criticized traditional software effort estimation models. They argue that software engineering data sets do not adhere to the parametric assumptions and traditional software effort estimation models do not provide any support for risk assessment and mitigation. They propose a Bayesian approach to rectify the problems posed by traditional software effort estimation models.

Even though Chulani et al. [9] and Fenton et al. [12], [13] make a compelling case for the use of the Bayesian model, they do not compare the performance of the Bayesian model with other nonlinear and nonparametric forecasting models such as neural network and regression tree models. Pendharkar and Subramanian [31] and Pendharkar [29] have used neural network and regression tree forecasting models to forecast software development effort and software size, respectively. While all three models, the Bayesian, the neural network, and the regression tree, may be appropriate to forecast software development effort, they are very different in their outputs and flexibility. The neural network models provide a point forecast of software development effort and do not provide any probabilities or certainty that the forecast software development effort may be achieved. Linear regression and genetic programming regression models have the same problem as well. The regression tree model called the classification and regression tree (CART) also provides a point forecast. But, given that a regression tree is provided as an output, a decision-maker may sometimes be able to compute an upper and a lower bound on the forecast, although it is not always possible to compute such upper and lower bounds [29]. The output of the Bayesian model is a joint probability distribution and not a point forecast. However, for the Bayesian model, a point forecast value might be computed from the joint distribution. The Bayesian model offers other features that make it attractive for the purpose of software development effort estimation. Heckerman [19] highlights four strengths of the Bayesian model. These four strengths of the Bayesian model for data mining are

- capability of handling missing data,
- capability of learning causal relationships,
- capability of combining prior knowledge and data, and
- capability of avoiding overfitting the data.

Despite the attractiveness of the Bayesian model for predicting software effort, only a few studies have used the Bayesian model for predicting software development effort. We believe that any software effort estimate given by any forecasting model has a degree of uncertainty associated with it. Bayesian models, in addition to providing a forecast of the software development effort, provide the joint probability distribution. A decision-maker can use the joint probability distribution information to estimate the probability that the forecast development effort will not be achieved. Managers can estimate the lower and upper

- *P.C. Pendharkar and G.H. Subramanian are with Information Systems, School of Business Administration, Capital College, Pennsylvania State University, 777 W. Harrisburg Pike, Middletown, PA 17057. E-mail: {pxp19, ghs2}@psu.edu.*
- *J.A. Rodger is with MIS and Decision Sciences, Eberly College of Business & Information Technology, Indiana University of Pennsylvania, Indiana, PA 15705. E-mail: jrodger@iup.edu.*

Manuscript received 7 Oct. 2004; revised 4 May 2005; accepted 5 May 2005; published online 26 July 2005.
Recommended for acceptance by B. Littlewood.
For information on obtaining reprints of this article, please send e-mail to: tse@computer.org, and reference IEEECS Log Number TSE-0203-1004.

bounds for effort estimates that can be useful in project risk assessment [32]. Further, Bayesian models provide a capability of updating the probability distribution, which might be a very useful tool in various stages of systems development.

In addition to the probability distribution information, Bayesian models provide several procedures to update the probability distribution when new information becomes available. The new information can be information obtained from subjective expert estimates on variables, which are not available in the historical software engineering data. The capability of updating the probability distribution due to the availability of new information makes the Bayesian model suitable for software effort estimation. Software development is a dynamic process and a manager's beliefs are likely to change over the development life cycle. When new information becomes available over time, a manager may incorporate the new information into the Bayesian model and estimate a new probability distribution.

The objectives of this study are as follows: First, through the review of literature, we summarize different software effort estimation models, identify factors that may affect the software development effort, and propose a probabilistic model. Second, using a real-world data set, we empirically compare the Bayesian model with two nonparametric neural network and regression tree approaches. Third, we illustrate how new information can be combined in a Bayesian software effort estimation model. In the end, we provide a summary of our findings and conclusions.

2 Literature Review on Software Effort Estimation Techniques, Factors Impacting the Software Development Effort, and a Probabilistic Model for Software Development Effort

Software effort estimation is a key antecedent to software cost estimation. Software effort is defined by the equation $effort = people * time$ [11]. Software effort estimation techniques fall into four categories—empirical, regression, theory-based, and machine learning techniques. Empirical techniques include analogy, function points (FP), and rules of thumb [22]. Regression techniques use parametric and nonparametric forecasting models [17]. The theory-based techniques use the underlying theoretical considerations characterizing some aspects of software development processes [11]. Examples of theory-based techniques are the COCOMO and the SLIM model (see the Appendix for an overview). Machine Learning (ML) techniques for predicting software effort involve Artificial Neural Networks (ANNs), Classification and Regression Tree (CART), Case-based Reasoning (CBR), Genetic Algorithm (GA), Genetic programming (GP), and Rule Induction (RI) [7]. A recent study by Jorgensen [23] provides a detailed review of different studies on the software development effort.

Recently, the traditional effort estimation techniques have been criticized for not providing appropriate causal relationships for the prediction of software development effort [12], [14]. Among the criticisms of traditional software development effort estimation techniques are their lack of support for risk assessment and reduction, inability to combine empirical evidence and expert judgment, and inability to handle incomplete information [14]. Fenton et al. [13], criticizing the traditional software development effort estimation techniques, write "There is no longer any excuse for not building predictive models that can support effective risk management decisions. Bayesian networks can be used to construct models that encapsulate causal influences on a development effort." A few researchers used Bayesian approaches for predicting software development effort. For example, Moses and Clifford [26] proposed the use of a Bayesian statistical modeling approach to support effort estimation in small development companies. The proposed approach allows a decision-maker to encode estimates to improve the software effort prediction accuracy. The authors found the Bayesian inference makes effort estimation very useful in small companies. The Bayesian inference error distribution helps decision-makers compare and adjust their effort estimates to the actual system development effort and other estimator's estimates. Chulani et al. [9] used the Bayes theorem to combine prior expert judged information with data-driven information to empirically illustrate that the Bayesian approach outperforms the multiple regression approach. Stamelos et al. [34] illustrated how Bayesian belief networks can be used to support expert judgment for software cost estimation.

There is some evidence that software engineering practitioners have adopted Bayesian belief networks for software effort estimation. For example, a Bayesian network-based tool called MODIST was developed by a major grant from the European Commission and several industry partners (http://www.modist.org). Another tool, called Agena Risk, developed by Agena Ltd., allows software project risk management capabilities (Agena Ltd. at http://www.agena.co.uk/bbn_article/bbns.html).

There are a few causal models available for prediction of software development effort. For example, Wrigley and Dexter [37] and Chrysler [10] proposed a general model that causally predicts the software development effort throughout the system development life cycle. Wrigley and Dexter's model consists of three independent variables: system requirements size, personnel experience, and method and tools. Subramanian and Zarnich [35], Nesi and Querci [27], and Banker and Slaughter [2] report that software development effort depends on software development tools, software development methodology, software developers, experience with the development tools, and the project size and complexity. In reality, software development effort depends on several complex variables [2], [35] whose interrelationships are often not very clear.

Subramanian and Zarnich [35] and Pendharkar and Subramanian [31] used three variables as predictors of software effort. These three variables are software development methodology, software development CASE tools, and programmer CASE tool experience. The three independent variables used by Pendharkar and Subramanian [35] are a subset of the variables specified in the COCOMO II 2000 model [4]. Using the COCOMO II model variables and the studies of Subramanian and Zarnich [35] and Pendharkar and Subramanian [31], we propose the following causal

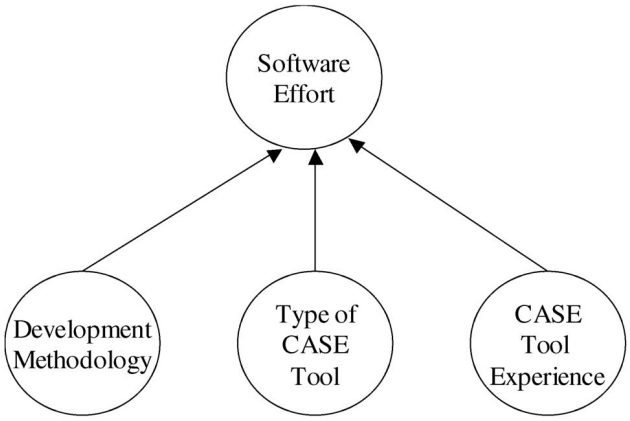


Fig. 1. A causal model for software effort prediction.

model for the prediction of software effort (Fig. 1). This causal model is limited to the three variables studied in [35], [31] and these variables are shown in Fig. 1. As mentioned earlier, we do understand that there are other complex variables that could also be studied if data on these variables are available.

We use Bayesian networks for learning joint probability distribution for the causal model illustrated in Fig. 1. Using the principles of the Bayesian networks, joint probability distribution of the causal model shown in Fig. 1 is given as follows:

$$P(Effort, Methodology, Tool\ Experience, Tool) = P(Effort|Methodology, Tool\ Experience, Tool) P(Methodology)P(Tool\ Experience)P(Tool).$$

Let $IV = \{Methodology, Tool\ Experience, Tool\}$ be the set of independent variables with its elements, represented as $iv_1 = Methodology$, taking one of the two possible values from its value set of

$$IVValue_1 = \{System\ Development\ life\ cycle\ (SDLC), Rapid\ Application\ Design\ (RAD)\},$$

$iv_2 = Tool\ Experience$ one of the three possible values from its value set of $IVValue_2 = \{low, medium, high\}$, and $iv_3 = Tool$ taking one of the two possible values of two computer aided software engineering (CASE) tools, Electronic Data Systems' INCASE tool, and Texas Instruments' Information Engineering Facility (IEF) case tool. Thus, $IVValue_3 = \{INCASE, IEF\}$. Let $DV = \{Effort\}$ be the set of dependent variable taking values from its value set $DVValue = \{low, medium, high\}$. Further, let $Examples = \{examples_1, \ldots, examples_n\}$ be the set of all the n historical examples in the database. Each element of the set of Examples consists of one example which can be represented as the union of all the elements of the set of IV and DV, with each element taking a certain allowable value from its value set. For example, an element i for some $i \in \{1, .., n\}$ may be as follows:

$$Element_i = \{Methodology = SDLC, Tool\ Experience = low, Tool = INCASE\}.$$

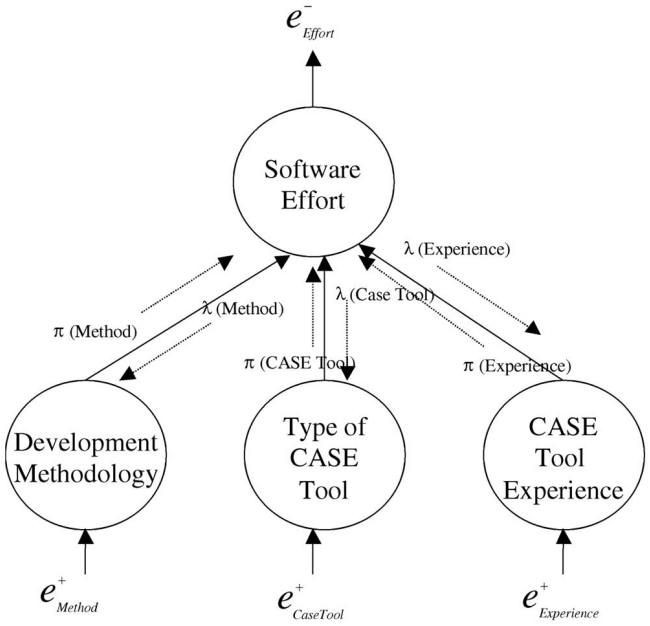


Fig. 2. Belief updating by network propagation in the Bayesian software effort model.

In our research, we use a standard naive Bayes network procedure to learn and predict software development effort. Although standard naive Bayes procedure allows managers to learn and forecast software development effort, it is not very adaptive. For example, Laranjeira [24] argues that initial software project estimates are rarely correct and the estimates can be improved during the project life cycle. One of the reasons why initial estimates are rarely correct may be due to the uncertainties about the selection of tools and techniques. The joint probability distribution formula described above is useful if a manager knows the tool type, methodology, and programmers' tool experience with certainty. However, in reality, a manager may be only partially sure about his resources and methodology. In order to model uncertainty in managerial decision-making, we use a belief updating procedure to complement the naive Bayes network.

As shown in Fig. 2, let e^+ and e^- represent new evidential data vectors at the parent and the leaf nodes. In Fig. 2, we use subscripts to highlight the type of evidence information. Further, let $\pi(iv_j)$ and $\lambda(iv_j)\ \forall j \in \{1, 2, 3\}$ represent the new prior and likelihood vectors in light of all the new evidential information vectors e^+ and e^-, and 1 represent vector of 1s of appropriate dimension, two or three in our case. Whenever new information becomes available, belief and prior updating procedures, described in Pearl [28], can be used to update the beliefs and priors.

3 Data, Experiments, and Results

For our experiments, we use a data set that has been used in previous studies in software engineering literature [35]. The data set used by Subramanian and Zarnich [35] consists of 40 software projects obtained from two major companies in the northeastern US. The data set includes an assigned project number (1 through 40), tool name (IEF or INCASE),

TABLE 1
The Prior Probabilities

Priors	Probability Value Ori. Data (33 Projects)	Probability Value Trn. Data (26 Projects)
P(Experience = Low)	0.606	*0.61*
P(Experience=Medium)	0.303	*0.31*
P(Experience=High)	0.091	*0.08*
P(Methodology=RAD)	0.212	*0.19*
P(Methodology=Non RAD)	0.788	*0.81*
P(Tool=IEF)	0.151	*0.19*
P(Tool=INCASE)	0.849	*0.81*

methodology (RAD or SDLC), tool experience (low, medium, or high), actual effort in man months, adjusted function points (AFP), unadjusted function points (UFP), and technical complexity factor (TCF). Since the original data set of 40 projects contains projects of different sizes, we eliminate projects with sizes greater than or close to 1,000 function points and use only 33 projects for our data analysis. The mean size (FP) and standard deviation of 33 projects were 223.7 and 155.01, respectively. The TCF values of projects did not vary much across the projects [mean = 0.905; S.D. = 0.12; max = 1.1; min = 0.7] which would suggest that the projects were comparable.

In defining the data, methodology and tool are self-explanatory. The methodology is defined as RAD or SDLC. The ICASE tool experience category is broken into three levels. This three level classification was chosen by us in consultation with the selected project managers from these organizations. The first level, low experience, is determined as no project member has over 1.5 years of experience utilizing the respective ICASE tool. The third level, high experience, is determined as at least one-half of the project members have over 3 years of experience utilizing the respective ICASE tool. The second level, medium experience, is determined as the project team falling somewhere between the low and high levels of experience.

Since there is no study that compares the performance of the Bayesian networks with other popular forecasting models, we benchmark the performance of the Bayesian networks with nonparametric neural networks and CART algorithm. Benchmarking the performance of nonparametric techniques is important as there are several commercial tools available to forecast software development effort. It is a very difficult task to select the best matching tool among several available commercial tools for a given problem. Since there is a lack of a theoretical framework that allows a decision-maker to select an appropriate forecasting model for a given forecasting problem, empirical studies provide the sole means for comparative analyses [3]. Using a previous study as a basis, we divide our original data set into 26 training and 7 test examples [31]. The neural network size and stopping criterion were similar to that of the Pendharkar and Subramanian [31] study that used the same data set. Our implementation of the CART algorithm is the same as that of the Pendharkar [29] study. The reader is directed to the Pendharkar and Subramanian [31] and Pendharkar [29] studies for further information on neural network, the CART algorithm, and their implementations. A brief introduction for the CART algorithm is provided in the Appendix.

Since software development effort is continuous and the Bayesian network required the use of discrete variables, we approximated the effort using three different discrete intervals. Varis [36] provides a methodology for discretization of continuous variables for Bayesian networks. Using the Varis [36] methodology and adjusting for sparse data, we used three discrete approximations. The three different approximations were as follows: Effort was categorized as low when the actual effort was less than or equal to 10 man-months. The mean value (μ_L) of *low* effort was 2.10 man-months. Similarly, the effort was categorized as *medium* when the actual effort was higher than 10 man-months and less than or equal to 20 man-months with a mean value (μ_M) of 14.50 man-months. Last, the effort was categorized as *high* when the actual effort was greater than 20 man-months with a mean value (μ_H) of 30.6 man-months. Tables 1 and 2 illustrate the values of prior and conditional probabilities, which were obtained by executing naive Bayes procedure. For Table 1, the first column indicates the values for the original data of 33 projects and the second column in italic font indicates the values for the training data of 26 projects. Table 2 illustrates the conditional probabilities for original and training data sets. The columns in Table 2 list the probability values of software effort conditioned on the known variables listed in the rows. The original and training data set values are listed into subcolumns of a given software development effort type column; the first subcolumn is the original data conditional probability. Since our data set is relatively small, we did not have enough projects to compute all conditional probabilities. Thus, in Table 2, the unknown conditional probabilities are represented by "-."

Our Bayesian network assumes that independent variables in the causal network are independent of each other [28]. Since our variables are binary/discrete, we test for associations among the variables using market-basket analysis. Market basket analysis is a popular data mining technique that tests for associations among binary variables. More information on market-basket analysis and the algorithm *Apriori* that is used for market-basket analysis can be found in Chen et al. [8].

Since market-basket analysis requires binary variables and the tool experience variable has three categories, we

TABLE 2
The Conditional Probabilities

(Method., Tool, Exp.)	Effort=High		Effort=Medium		Effort=Low	
(Non RAD, IEF, Low)	1.00	*1.0*	0	*0*	0	*0*
(RAD, IEF, Low)	0	*0*	1.0	*1.0*	0	*0*
(Non RAD, INCASE, Low)	0.06	*0.08*	0.06	*0.08*	0.88	*0.84*
(RAD, INCASE, Low)	-	-	-	-	-	-
(Non RAD, IEF, Medium)	0	*0*	1.0	*1.0*	0	*0*
(RAD, IEF, Medium)	-	-	-	-	-	-
(Non RAD, INCASE, Medium)	0	*0*	0	*0*	1.0	*1.0*
(RAD, INCASE, Medium)	0	*0*	0	*0*	1.0	*1.0*
(Non RAD, IEF, High)	-	-	-	-	-	-
(RAD, IEF, High)	-	-	-	-	-	-
(Non RAD, INCASE, High)	0	*0*	0.5	*1.0*	0.5	*0*
(RAD, INCASE, High)	-	-	-	-	-	-

& The sub-first column represents the conditional probability for original data of 33 projects and the second sub-column represents the conditional probability for training data of 26 projects.

combine categories low and medium into one category. The remaining two variables, methodology and tool, were binary and kept unchanged. Using the Apriori algorithm in the SPSS data mining package Clementine, we conducted the market-basket analysis to test the associations among the independent variables. For low support and threshold values of both equal to 0.6, we did not find any associations among the independent variables.

Since the Bayesian network provides a probability distribution over the effort as low, medium, or high, the results of the Bayesian network cannot be directly compared to the results of the neural network and the CART algorithm. In order to compute a point forecast, we suggest the following procedure, shown in Fig. 3, to compute normalized joint probability distribution over software development effort given certain values of development methodology, tool experience and type of CASE tool. Since the values of development methodology, tool experience, and type of CASE tool are known, the normalized joint probability distribution is the belief distribution [28].

Once the belief distribution of the software development effort is known, the point forecast for the software development effort can be computed using the following Lemma 1.

Lemma 1. *If $p = (p_L, p_M, p_H)$ is a normalized vector representing the probability distribution of development effort for a given methodology, tool, and tool experience and μ_L, μ_M, μ_H*

Procedure Compute Normalized Joint Distribution

Begin

$iv_1 \leftarrow Constant1 \in IVValue_1$
$iv_2 \leftarrow Constant2 \in IVValue_2$
$iv_3 \leftarrow Constant3 \in IVValue_3$

for $k \leftarrow 1$ *to* 3

do

$$P(dv_k, iv_1, iv_2, iv_3) = P(dv_k \mid iv_1, iv_2, iv_3) \prod_{j=1}^{3} P(iv_j)$$

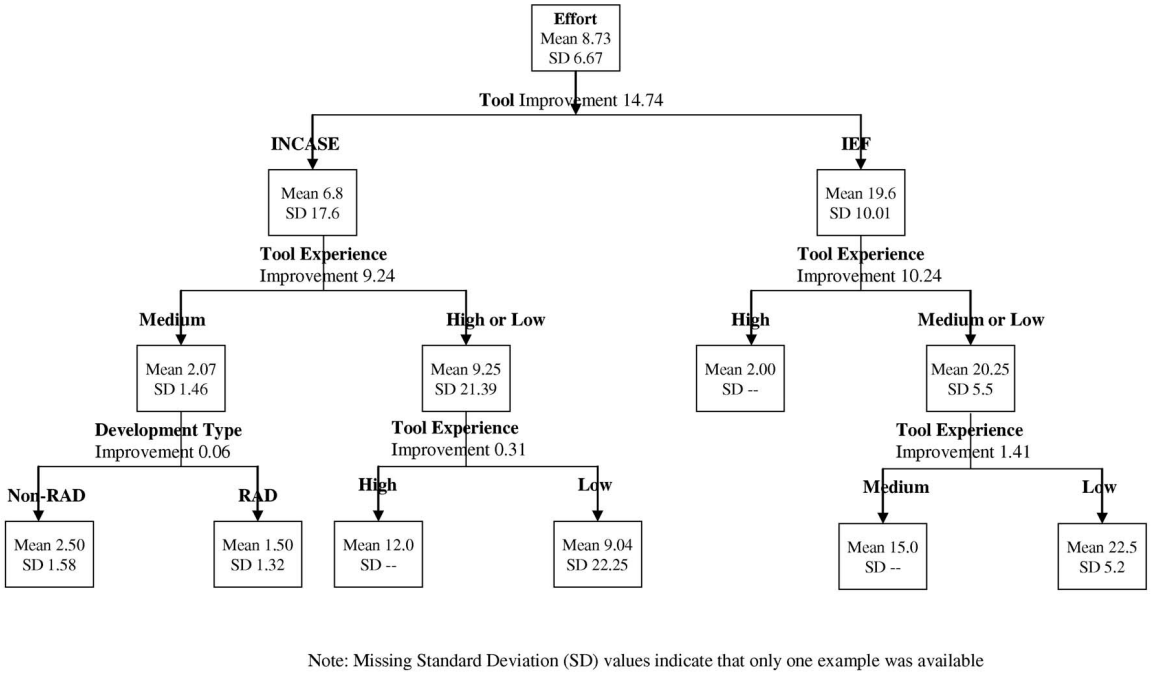
Enddo

$$Compute\ \alpha\ such\ that\ \alpha \sum_{k=1}^{3} P(dv_k, iv_1, iv_2, iv_3) = 1$$

$p_L \leftarrow \alpha P(dv_1, iv_1, iv_2, iv_3)$
$p_M \leftarrow \alpha P(dv_2, iv_1, iv_2, iv_3)$
$p_H \leftarrow \alpha P(dv_3, iv_1, iv_2, iv_3)$
$\boldsymbol{p} \leftarrow (p_L, p_M, p_H)$

Return p

Fig. 3. A procedure to compute joint distribution of software development effort.



Note: Missing Standard Deviation (SD) values indicate that only one example was available

Fig. 4. The CART regression tree for the training data.

are the mean values of low, medium, and high effort, then the estimated software development effort is given as follows:

$$E(Effort) = p_L\mu_L + p_M\mu_M + p_H\mu_H.$$

Proof. Using the expectation relation, the expected effort is given as,

$$E(Effort) = p_L E(Effort = low) + p_M E(Effort = Medium) + p_H E(Effort = High).$$

Substituting the mean values for the expected effort equal to low, medium, and high, we get the result. □

Using the point forecast for the Bayesian network, we compare the out-of-sample (test data) performance for the three forecasting techniques. We use the 26 software projects for training the techniques and the remaining seven projects for testing the performance of each technique. Fig. 4 illustrates the regression tree learnt by the CART algorithm on the training data. For leaves with missing standard deviation values, only one case in the training data belonged to that leaf. Fig. 4 illustrates that the type of CASE tool is a primary antecedent for predicting software development effort as the greatest improvement is obtained on the tool binary split. Fig. 5 illustrates the test performance of the three techniques on the test data set of seven projects. The bold line indicates the actual value of the effort. The acronym ANN indicates the (artificial) neural network technique.

Table 3 illustrates the pairwise difference in means between the actual effort and three different forecasting approaches and the pairwise difference in means between the three different forecasting approaches. No significant difference in means was observed between actual effort and the predicted effort using the Bayesian forecasting approach. However, the differences in means between

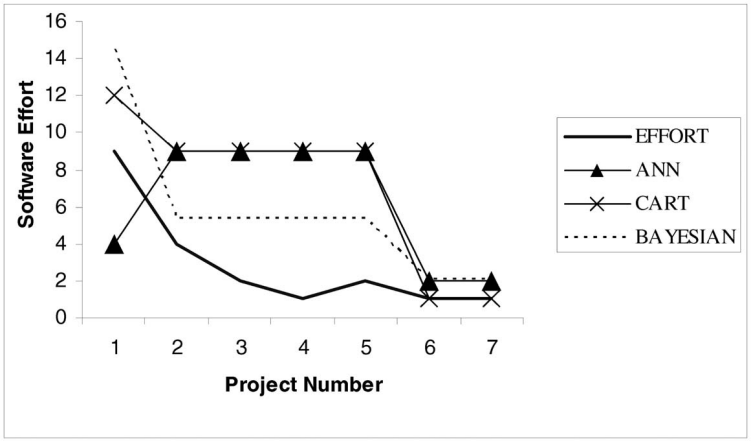


Fig. 5. The test data performance of the three forecasting techniques.

TABLE 3
Pairwise Difference in Means |t|-Test Results for the Test Data

Mean Effort	Mean ANN	Mean Bayesian	Mean CART	|t|-value	P < t
2.86	6.29			4.38	0.02*
2.86		5.74		1.93	0.05
2.86			7.14	3.38	0.01*
	6.29	5.74		0.27	0.390
	6.29		7.14	0.712	0.251
		5.74	7.14	1.31	0.117

* significant at 95% confidence

TABLE 4
The Absolute Values of Errors for Three Different Forecasting Techniques

Project number	Absolute Error for ANN	Absolute Error for Bayesian Network	Absolute Error for CART
1	5	5.5	3
2	5	1.37	5
3	7	3.37	7
4	8	4.37	8
5	7	3.37	7
6	1	1.1	0
7	1	1.1	0

TABLE 5
The Predicted Category and a Probabilistic Bound for the Software Effort

Project number	Actual Category	Predicted Category	P(Effort < 20)
1	Low	Medium	100%
2	Low	Low	92%
3	Low	Low	92%
4	Low	Low	92%
5	Low	Low	92%
6	Low	Low	100%
7	Low	Low	100%

actual and ANN, and actual and CART forecasting approaches were significant. The pairwise comparisons between all three forecasting techniques were nonsignificant, indicating that all three techniques behaved similarly. The mean absolute relative errors for ANN, Bayesian network, and CART were 2.68, 1.56, and 2.37, respectively. Table 4 illustrates the actual values of absolute errors for ANN, Bayesian network, and CART techniques. The results indicate that the Bayesian technique behaves similarly to ANN and CART models, but is slightly more accurate than the ANN and the CART forecasting models.

The Bayesian approach allows the manager to estimate probability bounds on the forecast software effort. For example, given vector p, the probability that the software effort will not be *high* is given by the sum of p_L and p_M. Table 5 illustrates the actual value and category of the software development effort in the test data, the probability that the effort will be less than 20 man-months (as calculated by the Bayesian joint distribution), and the predicted effort using the Bayes procedure. The results in Table 5 indicate that the software development effort category prediction accuracy of the Bayes procedure is about 86 percent.

The capability of generating a point forecast, categorical value of a forecast, and a probability bound for a forecast are only a few advantages that the Bayesian forecasting model can provide. Most researchers in the past have noted that the main strength of a Bayesian model is its capability of integrating uncertainties and managerial estimate into the posterior estimates. For example, let us assume that the development methodology, type of CASE tool, and the programmers' CASE tool experience is not known with certainty. If a manager believes there is a 60 percent chance that he/she may use RAD methodology, 80 percent chance that he/she may use INCASE tool, and 60 percent chance that the programmers' experience in CASE tool will be low and 20 percent chance that the experience may be medium, what is the expected software development effort and the uncertainty associated with it? Clearly, traditional forecasting models, such as neural network and CART, cannot provide the capability to handle uncertain information. However, the Bayesian model can incorporate this uncertainty.

We use the entire data set of 33 projects and belief and prior updating procedures [28] to illustrate how a Bayesian model can provide an answer to the above question. Based

on the given information, we identify the prior vectors as follows:

$$\boldsymbol{\pi}(Method) = (\pi(Method = RAD), \pi(Method = NON\_RAD)) = (0.6, 0.4),$$
$$\boldsymbol{\pi}(Tool) = (\pi(Tool = IEF), \pi(Tool = INCASE)) = (0.2, 0.8),$$
$$\text{and } \boldsymbol{\pi}(Experience) = (\pi(Experience = Low), \pi(Experience = Medium), \pi(Experience = High)) = (0.6, 0.2, 0.2).$$

Using the prior updating procedure, we initialize likelihood vectors as follows:

$$\boldsymbol{\lambda}(Method) = (1, 1),$$
$$\boldsymbol{\lambda}(Tool) = (1, 1), \text{ and}$$
$$\boldsymbol{\lambda}(Experience) = (1, 1, 1).$$

We then calculate the values of

$$\boldsymbol{\pi}(Effort) = (\pi(Effort = Low), \pi(Effort = Medium), \pi(Effort = High)).$$

The value for $\pi(Effort)$ is computed by the following equation:

$$\pi(Effort) = \sum_{iv_1, iv_2, iv_3} P(Effort \mid iv_1, iv_2, iv_3)\pi(iv_1)\pi(iv_2)\pi(iv_3).$$

Using the conditional probability values from Table 2, we get $\pi(Effort = Low) = 0.36096$. Similarly, the values for $\pi(Effort = Medium)$ and $\pi(Effort = High)$ are 0.13152 and 0.05952, respectively. We also have $\lambda(Effort) = (1, 1, 1)$. Finally, after accounting for the uncertainty, we compute the new belief distribution on effort as,

$$\boldsymbol{BEL}(Effort) = \alpha.(1, 1, 1).(0.39136, 0.13664, 0.072) = (0.65, 0.24, 0.11).$$

Using Lemma 1, the expected effort is 8.22 man-months, and there is an 89 percent chance that the actual effort may be less than 20 man-months. If a manager was certain to use RAD methodology, INCASE tool with medium programmer tool experience, then the expected effort would be 2.10 man-months, and the probability that the actual effort may be less than 20 man-months would be 100 percent.

The Bayesian model, in addition to allowing for uncertainty in input information, allows managers to combine external information that is not used by the Bayesian model to generate a forecast. For example, assume that, using variables not used in the Bayesian model, an independent forecast is available. This forecast, either using guesstimates or function points, suggests that the effort probability distribution is $\boldsymbol{\pi}(Effort) = (0.3, 0.5, 0.2)$. The Bayesian model allows a decision maker to combine this new information with the existing information to improve the belief distribution on effort and update the priors.

Using the belief updating procedure [28], the new information changes the likelihood $\boldsymbol{\lambda}(Effort) = (0.3, 0.5, 0.2)$. The $\boldsymbol{BEL}(Effort)$ changes as follows:

$$\boldsymbol{BEL}(Effort) = \alpha.(0.3, 0.5, 0.2).(0.65, 0.24, 0.11) = (0.58, 0.36, 0.06).$$

The expected effort will now be 8.28 man-months with 94 percent chance that the actual effort may be less than 20 man-months. We can compute new likelihood vectors $\boldsymbol{\lambda}(Method)$, $\boldsymbol{\lambda}(Tool)$, and $\boldsymbol{\lambda}(Experience)$ as described in Pearl [28]. For example, $\lambda(Method)$ an element of $\boldsymbol{\lambda}(Method)$ vector can be calculated using the following equality:

$$\lambda(Method) = \sum_{iv_2, iv_3} \pi(iv_2)\pi(iv_3)\left\{\sum_{Effort} P(Effort|Method, iv_2, iv_3)\lambda(Effort)\right\}.$$

Substituting the values in the above equality, we get $\lambda(Method = RAD) = 0.164$. Similarly,

$$\lambda(Method = NON\_RAD) = 0.303,$$
$$\lambda(Tool = IEF) = 0.268,$$
$$\lambda(Tool = INCASE) = 0.165,$$
$$\lambda(Experience = Low) = 0.174,$$
$$\lambda(Experience = Medium) = 0.280, \text{ and}$$
$$\lambda(Experience = High) = 0.128.$$

The new priors are as follows:

$$\boldsymbol{\pi}^{new}(Method) = .(0.164, 0.303).(0.6, 0.4) = (0.448, 0.552)$$
$$\boldsymbol{\pi}^{new}(Tool) = \alpha.(0.268, 0.165).(0.2, 0.8) = (0.289, 0.711)$$
$$\boldsymbol{\pi}^{new}(Experience) = \alpha.(0.174, 0.280, 0.128).(0.6, 0.2, 0.2) = (0.561, 0.301, 0.138).$$

4 Conclusions and Summary

There are several software effort forecasting models that can be used in forecasting future software development effort. The Bayesian model, when used for forecasting software effort, offers certain unique advantages. Like most other forecasting models, the Bayesian can be used to provide a point forecast for a software development effort. However, unlike most other forecasting models, the Bayesian model allows the managers to generate probability bounds on the forecast effort and combine managerial subjective estimates with the existing historical data to improve the software effort forecast. The Bayesian model is able to handle missing data and can be used to learn the causal relationships [18]. Thus, the primary advantage of using the Bayesian model over other models is its capability of providing uncertainty in forecasted value and its capability of handling missing data.

Our study contributes to the literature in several ways. First, using a set of real-world software projects, we benchmark the performance of Bayesian point software development forecasts with popular nonparametric neural network and the CART approaches. The results of our benchmarking indicate that the point forecasts generated by the Bayesian model are competitive. Second, we illustrate how a manager can establish probability bounds on the software effort forecast generated by the Bayesian model. Third, we illustrate how subjective managerial estimates (resulting from new out-of-data information) can be incorporated into the Bayesian model to update the probabilities in the Bayesian network.

The aforementioned contributions are all unique contributions of our study. The Bayesian model has been used in previous studies in the literature [9], [34]. The Chulani et al. [9] and Stamelos et al. [34] studies focused on combining subjective managerial estimates into the Bayesian model. Chulani et al. [9] did not use the Bayesian network, but used the Bayes theorem to generate posterior distribution given a prior distribution. Stamelos et al. [34] study did not provide the elaborate belief updating procedure, but illustrated how the Bayesian network can be applied to COCOMO cost factors and to deal with productivity in the enterprise resources planning (ERP) system localization. Thus, we believe that our study is much broader and more comprehensive than some of the previous studies in the literature and complements the findings of the previous studies.

Our study, while significant, can be improved in several ways. For example, in our project scenarios, we assumed that the information on the uncertainty in managerial decision-making was available. When this information is not available, several techniques can be used to obtain such information. One approach could be to use Analytic Hierarchy Process (AHP) to obtain the priors. The link between AHP and Bayesian probabilities is well established [33]. The other approach could be to use Delphi or guesstimates [21].

APPENDIX

SOFTWARE EFFORT ESTIMATION MODELS, THE CART ALGORITHM, AND PROOF OF COMPUTATIONAL COMPLEXITY PROPOSITIONS

A.1 COCOMO/COCOMO II

COCOMO was the most popular model for software cost estimation during the 1980s and 1990s [9]. COCOMO estimates effort for an application using the following model form:

$$E = a(EDSI)^b \times EAF.$$

In the COCOMO model, the variable E is an effort estimate in man-months, $EDSI$ is estimated delivered source instructions, EAF is the effort adjustment factor, and parameters a and b are constants determined by application complexity. COCOMO assumes the systems development life cycle (SDLC) approach to application development, and an application effort can be assessed in three different levels: basic, intermediate, and advanced. The basic level effort is determined early in SDLC, while the intermediate and advanced levels are applied at later stages in SDLC. Depending on the application level, the overall COCOMO remains the same, except EAF is determined as follows:

$$EAF = \begin{cases} 1 \text{ (for basic level)} \\ \prod_{i=1}^{15} C_i \text{ (for intermediate and advance level)}, \end{cases}$$

where C_i represents the value of ith cost factor. The COCOMO model has been very accurate in estimating effort of applications in the intermediate level. The effort estimations at the basic and advanced levels were not so accurate [20].

The COCOMO II model uses software size as a primary factor and 17 secondary factors, called cost drivers [5]. The COCOMO II model has the following mathematical form:

$$E = a \times \left[Size\right]^{1.01+\sum_{i=1}^{3} SF_i} \times \prod_{i=1}^{17} EM_i,$$

where E and a are the same as before, $Size$ is the size of the software project (either measured in KLOC or FP), SF is the scale factor, and EM is the effort multiplier (cost drivers).

A.2 The SLIM (Software Lifecycle Management) Method

The SLIM method is a theory-based technique to estimate software development effort and schedule. The SLIM method is based on Putman's life-cycle analysis of the Rayleigh distribution. The Rayleigh distribution is used to estimate how many people should be allocated to the development and maintenance cost during the life cycle of a software project. The SLIM model uses two equations: the software productivity level equation and the manpower equation [11]. The software productivity level equation expresses development effort in terms of project size and development time. The manpower equation expresses the buildup of manpower as a function of time. The SLIM model uses the Rayleigh distribution to estimate project effort schedule and defect rate. Two key variables used in the SLIM method are the Productivity Index (PI) and the Manpower Buildup Index (MBI). The PI is a measure of process efficiency (cost-effectiveness of assets), and the MBI determines the effects on total project effort that result from variations in the development schedule.

A.3 Classification and Regression Tree Algorithm

Classification and Regression Trees (CART) is a binary decision tree algorithm that is used in data mining problems involving classification and regression. The CART constructs a binary decision tree by splitting a data set in such a way that the data in the descendant subsets are more *pure* than the data in the parent set. For example, in a regression problem, let (x_n, y_n) represent nth example, where x_n is the nth example vector on independent variables and y_n is the value of the dependent variable. If there are a total of N examples, then CART calculates a best split s^* so that the following is maximized over all possible splits S:

$$\Delta R(s^*, t) = \underset{\mathrm{s}\in\mathrm{S}}{\operatorname{argmax}} \Delta R(s, t),$$

where $\Delta R(s,t) = R(t) - R(t_L) - R(t_R)$ is improvement in the resubstitution estimate for split s of t. The resubstitution estimate $R(t)$ is defined as follows:

$$R(t) = \frac{1}{N} \sum_{\mathrm{x_n}\in t} (y_n - y(t))^2.$$

The variables t_L and t_R are left and right values for split t. The variable $y(t)$ is defined as follows:

$$y(t) = \frac{1}{N(t)} \sum_{x_n \in t} y_n,$$

where $N(t)$ is the total number of cases in t. The tree continues to grow until a node is reached such that no significant decrease in resubstitution estimate is possible. This node is the terminal node.

REFERENCES

[1] J. Baik, B. Boehm, and B.M. Steece, "Disaggregating and Calibrating the CASE Tool Variable in COCOMOII," *IEEE Trans. Software Eng.,* vol. 28, no. 11, pp. 1009-1022, Nov. 2002.

[2] R.D. Banker and S.A. Slaughter, "A Field Study of Scale Economies in Software Maintenance," *Management Science,* vol. 43, no. 12, pp. 1709-1725, 1997.

[3] S. Bhattacharyya and P.C. Pendharkar, "Inductive, Evolutionary, and Neural Computing Techniques for Discrimination: A Comparative Study," *Decision Sciences,* vol. 24, no. 4, pp. 871-899, 1998.

[4] B. Boehm, C. Abts, A. Brown, S. Chulani, B. Clark, and E. Horowitz, *Software Cost Estimation with COCOMO II.* Prentice Hall, 2001.

[5] B.W. Boehm, B. Clark, C. Horowitz, C. Westland, R. Madachy, and R. Selby, "Cost Models for Future Software Life Cycle Processes: COCOMO 2.0.0," *Annals of Software Eng.,* vol. 1, no. 1, pp. 1-30, 1995.

[6] L. Breiman, J.H. Friedman, R.A. Olshen, and C.J. Stone, *Classification and Regression Trees.* Belmont, Calif.: Wadsworth Int'l Group, 1984.

[7] C.J. Burgess and L. Lefley, "Can Genetic Programming Improve Software Effort Estimation? A Comparative Evaluation," *Information and Software Technology,* vol. 43, no. 14, pp. 863-873, 2001.

[8] M.S. Chen, J. Han, and P.S. Yu, "Data Mining: An Overview from a Database Perspective," *IEEE Trans. Knowledge and Data Eng.,* vol. 8, no. 6, pp. 866-883, Dec. 1996.

[9] S. Chulani, B. Boehm, and B. Steece, "Bayesian Analysis of Empirical Software Engineering Cost Models," *IEEE Trans. Software Eng.,* vol. 25, no. 4, pp. 573-583, July/Aug. 1999.

[10] E. Chrysler, "Some Basic Determinants of Computer Programming Productivity," *Comm. ACM,* vol. 21, no. 6, pp. 472-483, 1978.

[11] R.E. Fairley, "Recent Advances in Software Estimation Techniques," *Proc. Int'l Conf. Software Eng.,* pp. 382-391, 1992.

[12] N.E. Fenton, W. Marsh, M. Neil, P. Cates, S. Forey, and M. Tailor, "Making Resource Decisions for Software Projects," *Proc. 26th Int'l Conf. Software Eng.,* pp. 397-406, 2004.

[13] N.E. Fenton, P. Krause, and M. Neil, "Software Measurement: Uncertainty and Causal Modeling," *IEEE Software,* vol. 10, no. 4, pp. 116-122, July/Aug. 2002.

[14] N.E. Fenton and M. Neil, "Software Metrics: Roadmap," *The Future of Software Eng.,* A. Finkelstein, ed., pp. 357-370, 2000.

[15] N.E. Fenton and M. Neil, "Software Metrics: Successes, Failures and New Directions," *J. Systems and Software,* vol. 47, nos. 2-3, pp. 149-157, 1999.

[16] N. Fenton and S. Pfleeger, *Software Metrics: A Rigorous & Practical Approach.* PWS Publishing, 1997.

[17] A.R. Gray, S.G. MacDonnel, and M.J. Shepperd, "Factors Systematically Associated with Errors in Subjective Estimates of Software Development Effort: the Stability of Expert Judgment," *Proc. Sixth Int'l Software Metrics Symp.,* pp. 216-227, 1991.

[18] T.E. Hastings and A.S.M. Sajeev, "A Vector-Based Approach to Software Size Measurement and Effort Estimation," *IEEE Trans. Software Eng.,* vol. 27, no. 4, pp. 337-350, 2001.

[19] D. Heckerman, "Bayesian Networks for Data Mining," *Data Mining and Knowledge Discovery,* vol. 1, pp. 79-119, 1997.

[20] Q. Hu, R. Plant, and D. Hertz, "Software Cost Estimation Using Economic Production Models," *J. Management Information Systems,* vol. 15, no. 1, pp. 143-163, 1998.

[21] P.M. Johnson, C.A. Moore, J.A. Dane, and R.S. Brewer, "Empirically Guided Software Effort Guesstimation," *IEEE Software,* pp. 51-56, 2000.

[22] C. Jones, "By Popular Demand: Software Estimating Rules of Thumb," *Computer,* vol. 29, no. 3, p. 116, Mar. 1996.

[23] M. Jørgensen, "A Review of Studies on Expert Estimation of Software Development Effort," *J. Systems and Software,* vol. 70, pp. 37-60, 2004.

[24] L.A. Laranjeira, "Software Size Estimation of Object-Oriented Systems," *IEEE Trans. Software Eng.,* vol. 16, no. 5, pp. 510-522, May 1990.

[25] M.A. Mahmood, K.J. Pettingell, and A.I. Shaskevich, "Measuring Productivity of Software Projects: A Data Envelopment Analysis Approach," *Decision Sciences,* vol. 27, no. 1, pp. 57-80, 1996.

[26] J. Moses and J. Clifford, "Learning How to Improve Effort Estimation in Small Software Development Companies," *Proc. 24th Ann. Int'l Computer Software and Applications Conf. (COMPSAC),* pp. 522-527, 2000.

[27] P. Nesi and T. Querci, "Effort Estimation and Prediction of Object-oriented Systems," *J. Systems and Software,* vol. 42, no. 1, pp. 89-102, 1998.

[28] J. Pearl, *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference.* San Francisco, Calif.: Morgan-Kaufman, 1988.

[29] P.C. Pendharkar, "An Exploratory Study of Object-Oriented Software Component Size Determinants and the Application of Regression Tree Forecasting Models," *Information and Management,* vol. 42, no. 1, pp. 61-73, 2004.

[30] P.C. Pendharkar and S. Nanda, "A Misclassification Cost Minimizing Evolutionary-Neural Classification Approach," *Working Paper Series,* Working paper #03-6, School of Business Administration, Pennsylvania State Univ. at Harrisburg, 2004.

[31] P.C. Pendharkar and G.H. Subramanian, "Connectionist Models for Learning, Discovering, and Forecasting Software Effort: An Empirical Study," *J. Computer Information Systems,* vol. 43, no. 1, pp. 7-14, 2002.

[32] R.S. Pressman, *Software Engineering: A Practitioner's Approach.* McGraw-Hill, 2001.

[33] T.L. Saaty and L.G. Vargas, "Diagnosis with Dependent Symptoms: Bayes Theorem and the Analytic Hierarchy Process," *Operations Research,* vol. 46, no. 4, pp. 491-502, 1998.

[34] I. Stamelos, L. Angelis, and E. Sakellaris, "On the Use of Bayesian Belief Networks for the Prediction of Software Productivity," *Information and Software Technology,* vol. 45, no. 1, pp. 51-60, 2003.

[35] G.H. Subramanian and G. Zarnich, "An Examination of Some Software Development Effort and Productivity Determinants in ICASE Tool Projects," *J. Management Information Systems,* vol. 12, no. 4, pp. 143-160, 1996.

[36] O. Varis, "A Belief Network Approach to Optimization and Parameter Estimation: Application to Resource and Environmental Management," *Artificial Intelligence,* vol. 101, no. 1-2, pp. 135-163, 1998.

[37] C.D. Wrigley and A.S. Dexter, "A Model of Measuring Information System Size," *MIS Quaterly,* vol. 15, no. 2, pp. 245-257, 1991.

Parag C. Pendharkar is an associate professor of information systems at Pennsylvania State University, Harrisburg. He has published several papers in journals affiliated with ACM, IEEE, INFORMS, Decision Sciences Institute, John Wiley, and Elsevier. Currently, he serves as an associate editor for the *International Journal of Human-Computer Studies*.

Girish H. Subramanian is an associate professor of information systems at Pennsylvania State University, Harrisburg. His work has appeared in *Communications of the ACM*, *Decision Sciences*, *Journal of Management Information Systems*, *Journal of Systems & Software*, *Journal of Computer Information Systems*, as well as several other journals.

James A. Rodger is a professor of management information systems at Indiana University of Pennsylvania. He has published several journal articles in *Annals of Operations Research*, *Communications of the ACM*, *Computers & Operations Research*, *Decision Support Systems*, *Expert Systems with Applications*, *International Journal of Human-Computer Studies*, as well as several other journals.