A Bayesian Belief Network for Assessing the likelihood of the fault content

Sousuke AMASAKI, Yasunari TAKAGI, Osamu MIZUNO, and Tohru KIKUNO
Graduate School of Information Science and Technology, Osaka University
1–3 Machikaneyama, Toyonaka-shi, Osaka 560–8531, Japan
{amasaki,y-takagi,o-mizuno,kikuno}@ist.osaka-u.ac.jp

Abstract

For predicting the software quality, we must consider various factors because the software development consists of various activities, which Software reliability growth model (SRGM) does not consider.

In this paper, we propose a model to predict the final quality of software product by using the Bayesian belief network (BBN) model. By using the BBN, we can construct a prediction model that focuses on the structure of software development process explicitly, represents complex relationships between metrics, and handles uncertain metrics such as residual faults in software products. In order to evaluate the constructed model, we perform an empirical experiment based on the metrics data collected from the development projects in a certain company. As the result of the empirical evaluation, we confirmed that the proposed model can predict the amount of residual faults that the SRGM cannot handle.

Keywords: *Bayesian belief network, causal model, software quality prediction*

1 Introduction

In order to assure the software quality effectively, the prediction of software quality has become important. Various methods for predicting the software quality have been proposed so far. Software reliability growth model (SRGM) is one of the best-known methods, which is designed for software testing phase. Generally, the testing phase consists of the four activities: unit testing, integration testing, function testing, and system testing [10, 15]. In the SRGM, the change of the number of faults detected in each testing activity is represented by such a successive curve function that aims at fitting to the actual number of detected faults.

The software release time, that is the time to finish testing, can be decided using the ratio of the time to be spent and the number of faults expected to be detected, both of which are known from the curve. Thus, developers in actual companies have extensively applied the SRGM to decide the release time in order to manage testing efficiently.

In a certain company also, the SRGM has been used to decide the release time of a testing activity. In the most projects, the final quality of software products has been successfully assured to be good by the testing based on the SRGM. In other words, by applying the SRGM, the residual faults of software products is kept small. However, there remain such cases that the final quality of software products is still poor even after decision making by the SRGM. Therefore, the SEPG (Software Engineering Process Group) in the company thinks that the SRGM cannot give a precise prediction of the residual faults.

By discussions with SEPG, we conclude that the metrics data collected in the activities (before test activities) give a hint to solve the problem remained by the SRGM. The main reason of our conclusion is the fact that review activity affects the software quality [17].

The objective of our research is to find such risky projects (producing software products with the poor final quality), that cannot be detected by the SRGM, using metrics collected from all activities in the software development process.

As a method for predicting risky projects by using several metrics, the regression models are often used [3, 12]. However, the regression models have the following problems for our purpose [7].

First, the metrics that are highly correlated cannot be used in the regression model simultaneously. It is possible to avoid this difficulty by introducing a new composite metric of these correlated metrics. However, we want to use values of metrics directly, since it makes easy to investigate the relationship among metrics.

Second, the regression model cannot be applied when values of explanatory variable in the model is unknown. That is, when we try to use the regression model, all values of explanatory variable must be collected. The metrics, which affect the final quality of software, include the ones that are related to phases after the end of developmental activities, such as the number of detected faults after shipping.

Thus, in order to use such metrics in our research work, it is difficult to apply the regression model.

In our research, we try to apply Bayesian belief network (BBN) [4] as a modeling method to find risky project. The BBN is one of methods for modeling systems that include causal relationships among variables. The BBN can handle uncertainties as probabilistic events, and thus it can be extensively applied under the condition that not all values of metrics or variables are given. That is, even if we cannot collect all data corresponding to the metrics in the model, we can make decision or diagnose by using the probabilities for metrics which value is unknown. This feature of the BBN is useful for solving two problems of the regression model just mentioned.

In this paper, by using the BBN model, we firstly try to construct the prediction model for software quality. Concretely, by measuring the software quality as the amount of residual faults in the software product, the prediction model observes the changes of the amount of residual faults during the successive development activities. Next, by applying the actual project data, we evaluate the usefulness of prediction model. Here, in all of these project data, the SRGM is used to decide the release time, but unfortunately a few projects still have some residual faults in the final products. As the result of evaluation, we can say that the proposed model identifies the risky projects successfully, and that the model is applicable for complementing the SRGM.

The rest of this paper is organized as follows: Section 2 explains the BBN with example model. Section 3 describes the development process of the target project. In Section 4, we construct the prediction model for software quality using the BBN. We then evaluate the model by applying the actual project data in Section 5. Finally, Section 6 concludes this paper and shows the future works.

2 Bayesian Belief Network

2.1 Calculation using BBN

The Bayesian belief network (BBN) consists of two components: the graph and the probability table. The graph represents causal relationships between variables by the directed links connecting variables [4]. Formally, this graph is in the form of DAG (directed acyclic graph).

An example of DAG is shown in Figure 1. Since variable v_4 depends on variable v_2 and v_3, there exist links (v_2, v_4), (v_3, v_4) in DAG. Figure 1 implies that variable v_5 depends on variable v_1 and v_4 directly and variable v_2 and v_3 indirectly. In this case, we call v_5 as dependent variable.

On the other hand, the probability table is assigned to each variable. Especially, the probability table assigned to the dependent variable (here, v_4 and v_5 are such dependent variables) is named conditional probability table (CPT).

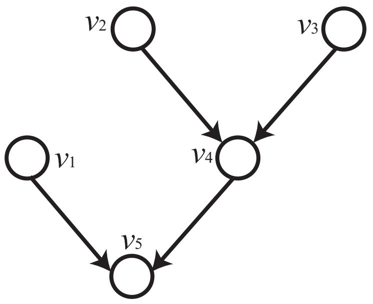


Figure 1. Example of DAG

Table 1. Conditional probability table for v_4

		v_4	
v_2	v_3	T	F
T	T	0.4	0.6
	F	0.3	0.7
F	T	0.6	0.4
	F	0.5	0.5

Table 1 is an example of CPT assigned to variable v_4, that shows the relationship among v_2, v_3, and v_4. Here, we assume that all variables are binary that take values “T” or “F”.

With the BBN, we can perform the calculation of the probability. There exist two typical cases:

Case 1: Values of all variables are known (Figure 2).

Case 2: Values of some variables are unknown (Figure 3).

In Figure 2, a black circle ● represents the node for which the value (that is, T or F) is assigned and a double circle ◎ represents the target node for which the probability must be calculated. For example, we calculate the probability of $v_5 = T$ under the condition that $v_1 = T$, $v_2 = T$, $v_3 = T$, and $v_4 = T$. We assume that the CPT for v_5 (shown in Table 2) is assigned to v_5. Because the value of v_4 is set to “T”, we don’t care about v_2 and v_3. Then, $p(v_5 = T|v_1 = T, v_2 = T, v_3 = T, v_4 = T) = p(v_5 = T|v_1 = T, v_4 = T) = 0.7$ is obtained from Table 2.

Next, in Figure 3, white circle ○ represents the node for which the value is not assigned and thus it is unknown. In order to calculate the probability of v_5, we need the value of v_4. However, in this case, the value of v_4 is unknown.

Before calculation, we assume that the CPT shown in Table 2 is assigned for v_5 and that the prior probability tables shown in Table 3, 4, and 5 are assigned for v_1, v_2, and v_3, respectively. Here, the prior probability table represents the distribution of the probability.

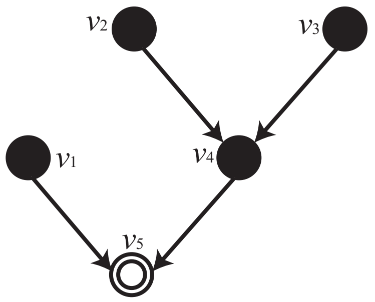


Figure 2. Assignment for Case 1

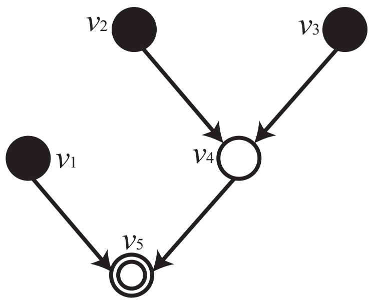


Figure 3. Assignment for Case 2

Here again, we calculate the probability of $v_5 = T$ under the condition that $v_1 = v_2 = v_3 = T$. First, from Table 1, we can get $p(v_4 = T|v_2 = T, v_3 = T) = 0.4$ and $p(v_4 = F|v_2 = T, v_3 = T) = 0.6$. If the value of v_4 were to be assigned, we would be able to determine the probability of $v_5 = T$ simply from Table 2. Although v_4 is uncertain, we can calculate the probability of $v_5 = T$ by using Table 2 as follows:

$$\begin{aligned}
&p(v_5 = T|v_1 = T, v_2 = T, v_3 = T)\\
&= p(v_5 = T|v_1 = T, v_4 = T)\\
&\qquad \times p(v_4 = T|v_2 = T, v_3 = T)\\
&\quad + p(v_5 = T|v_1 = T, v_4 = F)\\
&\qquad \times p(v_4 = F|v_2 = T, v_3 = T)\\
&= 0.7 \times 0.4 + 0.3 \times 0.6\\
&= 0.46
\end{aligned}$$

2.2 Application to Software Engineering

The BBN is used to deal with causal relationships among variables that allow uncertainty for some variables. In recent years, because of the capability of fast computation, the BBN is used in the widespread areas [2, 9, 11, 13] such as artificial intelligence, medical diagnosis, trouble shooting diagnosis, and decision making.

Table 2. Conditional probability table for v_5

		v_5	
v_1	v_4	T	F
T	T	0.7	0.3
	F	0.3	0.7
F	T	0.8	0.2
	F	0.3	0.7

Table 3. Prior probability table for v_1

v_1	
T	F
0.8	0.2

The BBN is introduced to the software engineering by Fenton [6–8]. In [8], they considered the BBN as a tool for the risk analysis and management. They modeled the defects using the BBN and analyze the relationships under the uncertainty. In this paper, we try to predict the software quality using the BBN model and perform the empirical evaluation statistically by applying the constructed model to the actual projects data.

3 Target Projects

3.1 Characteristics of Projects

The projects targeted in this paper are the development of computer control systems with embedded software in a certain company. The software products developed by the projects have the following common characteristics. The systems are related to retail systems, and thus embedded software implements rather complex functions dealing with many sensors, actuators and control signals including various kinds of interrupts. Furthermore, since it is delivered in the form of LSI chips, modification of faults after delivery is very expensive. Thus, high quality is especially required for the embedded software.

We use two datasets **DATA1** and **DATA2** collected from two project groups. **DATA1** consists of the actual project data of 51 projects, which have already finished their development. Each project started its development from 1995 to 1998. **DATA2** consists of the actual project data of 47 projects, which have already finished their development, too. Each project started from 2001 to 2002.

3.2 Actual Development Process

In the target projects, the products are developed under a development process as shown in Figure 4. The devel-

Table 4. Prior probability table for v_2

v_2	
T	F
0.6	0.4

Table 5. Prior probability table for v_3

v_3	
T	F
0.7	0.3

opment process is an ordinal waterfall model. This process consists of three successive phases, design & coding phase, test & debug phase, and acceptance test & debug. The design & coding phase is divided into five activities: Concept design, Function design, Structure design, Module design, and Coding. On the other hand, the test & debug phase is divided into four activities: Unit test & debug, Integration test & debug, Function test & debug, and System test & debug. In the company, the software test is carried out in the early two activities, and the test on a hardware is carried out in the last two activities.

One characteristic of the design & coding phase is that review activity is introduced after each design activity. Review activity enables the detection and correction of faults in software artifacts as soon as these artifacts are created [1, 5]. The review activity not only improves the quality of the artifacts but also helps software development organizations to reduce their cost of producing software. In the review activity, the documents should be distributed to the persons concerned in the company, and then review results should be returned to developers via manager (this review activity is called peer review [1]). The SEPG in the company established several guidelines for the review activity. One of them directs that at least 15% of the total efforts for design & coding phase should be assigned to review activities [18].

The test & debug phase consists of the repetition of a pair of test activity and debug activity. Test activity is the process of analyzing a software item to detect the differences between existing and required conditions and to evaluate the features of the software items. The SRGM has been used to decide the release time of each testing activity. Debug activity is the process to detect, locate, and correct faults [14]. The persons engaged in the test and debug phase are directed to record all faults that are detected by the test activity and removed by the debug activity [19].

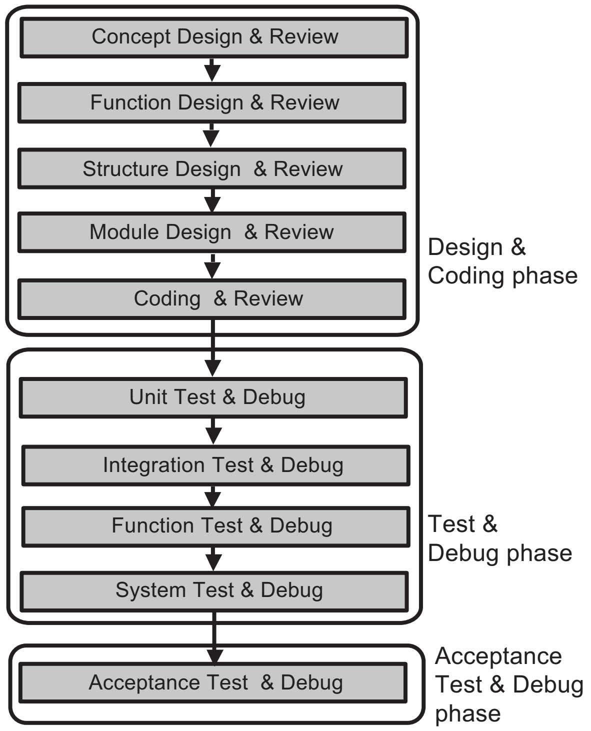


Figure 4. Development process

4 BBN Model for Quality Prediction

4.1 Software Metrics

(A) Abstract development process

In subsection 3.2, we describe the precise software process model. However, in order to construct the prediction model using the BBN for software quality, we summarize the software process. That is, we merge successive activities in Figure 4 into a new activity. Then we obtain the abstract development model shown in Figure 5.

- Design & review phase
  In the abstract development model, design & review phase consists of the one pair of design activity and review activity as shown in Figure 5. Here, the value of metric of a new activity is defined as the sum of the values of the corresponding old activities in Figure 4.

- Test & debug phase
  Precisely speaking, test & debug phase in Figure 4 consists of the test of developed software and the test of software integrated with hardware. The former test is carried out in unit and integration test & debug activities. On the other hand, the latter test is carried out in function and system test & debug activities. Then we

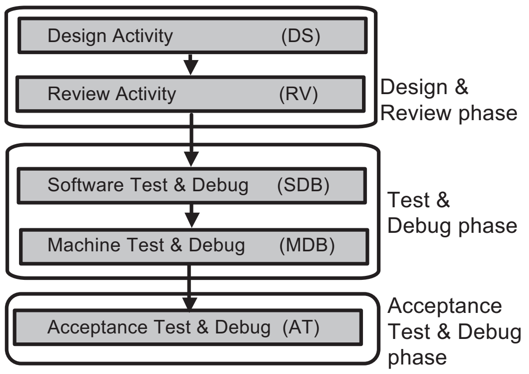

Design Activity (DS)
Review Activity (RV)
Design & Review phase
Software Test & Debug (SDB)
Machine Test & Debug (MDB)
Test & Debug phase
Acceptance Test & Debug (AT)
Acceptance Test & Debug phase

Figure 5. Abstract development process

merge unit and integration test & debug activities into software test & debug activity in Figure 5. Similarly, we merge function and system test & debug activities into machine test & debug activity.

- Acceptance test & debug phase
  Obviously from Figure 4, this phase consists of a single activity. Then this phase remains unmodified in Figure 5.

(B) Collection of Metrics

The metrics used in the proposed model are classified into five groups: S_α, E_α, DF_α, TI_α, and RF_α where α denotes an activity in Figure 5. The detailed definitions are given in the followings:

- Product size (Kstep): S_{DS}
- Effort (person-day): E_{DS}, E_{RV}, $E_{DB}(= E_{SDB} + E_{MDB})$, E_{AT}
- Detected faults (number): DF_{RV}, DF_{SDB}, DF_{MDB}, DF_{AT}
- Test Items (number): TI_{SDB}, TI_{MDB}, TI_{AT}
- Residual faults (number): RF_{DS}, RF_{RV}, RF_{SDB}, RF_{MDB}, RF_{AT}

In the above list, product size S_α, effort E_α, test items TI_α, and detected faults DF_α are observable in the sense that these are recorded in the actual software development projects. Precisely speaking, for $\alpha = DS, RV, SDB$, and MDB, the values of metrics S_α, E_α, DF_α, and TI_α are calculated by summing up the values at the corresponding activities. For example, effort E_{DS} is calculated as the total sum of efforts at concept design, function design, structure design, module design, and coding. On the other hand, we cannot record the metrics RF_α in each activity α because the number of injected faults cannot be counted a priori.

When we try to predict the software quality, the human factor such as the ability, the expertise, etc. should be considered as one of the key metrics for the prediction. However, these are not recorded, and thus we do not consider them in this paper. It is still remained as one of important future works.

(C) Calculation of RF's

It is a serious problem for the model construction that amounts of residual faults of the product RF_{DS}, RF_{RV}, RF_{SDB}, RF_{MDB}, and RF_{AT} are unknown.

In order to overcome this deficiency, we take the following approach. At first, we assume that RF_{AT} is equal to the number of detected faults during six months after shipping. Then, we can define the total number of faults RF_{DS} (that is, the residual faults in the design activity) to be the sum of RF_{AT} and the total number of faults detected in each activity of the software development process. Thus, we assume the following equation:

$$RF_{DS} = RF_{AT} + DF_{RV} + DF_{SDB} + DF_{MDB} + DF_{AT}$$

According to this definition, the residual faults RF_α after each activity α is calculated by subtracting the detected faults DF_α in each activity α from the residual faults in the previous activity.

4.2 Approach to Model Construction

Based on the abstract model in subsection 4.1, we construct the prediction model for software quality using Bayesian belief network. For software quality, we should consider various facet of a software because it consists of various criteria. However, especially in testing phase, software quality is verified how much a software is in keeping with a specification. Then, the smaller inconsistency between the software and the specification is, the higher the software quality is. From such viewpoint, the amount of residual faults in the product, which turns out the inconsistency, is important factor of software quality. In this paper, we regard the software quality as the amount of residual faults in the product. Then, we determine the following approach to model construction:

(1) Observe carefully the change in the amount of residual faults at each activity and represent it as the main flow of the model.

(2) Calculate the amount of residual faults in each phase from the metrics recorded at activities in the same phase.

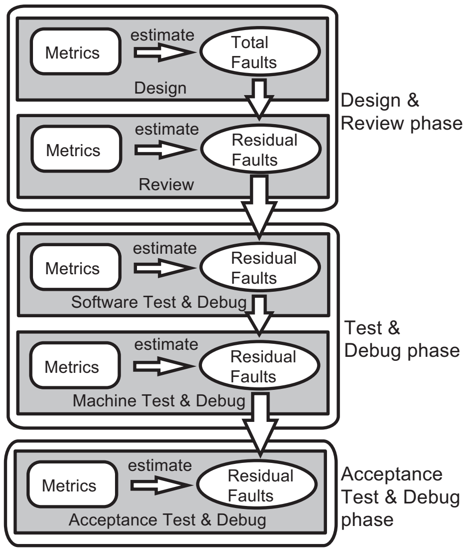


Figure 6. Overview of the prediction model

According to this approach, we construct the overview of prediction model shown in Figure 6.

Then, we consider that a final result of prediction by this model is the amount of residual faults in the acceptance test & debug activity.

4.3 Construction of DAG

We define a prediction model using the BBN, according to Figure 6, for software quality. First, we construct five kinds of DAGs for five activities: DS, RV, SDB, MDB, and AT shown in Figure 5. Then, we integrate these five DAGs into a DAG that represents the whole software development process.

(1) DAG for DS Here, we assume that faults are introduced by design activity, and that residual faults are removed by other activities such as review, test & debug in Figure 6. We assume that metrics S_{DS} and E_{DS} in design activity affect the number of the faults introduced RF_{DS}. The larger the design effort E_{DS} is, the larger the number of introduced faults RF_{DS} is. Similarly, the RF_{DS} becomes large in proportion to the size of the product S_{DS}. As a result, DAG shown in Figure 7 is obtained.

(2) DAG for RV Review is carried out in order to remove faults remained or introduced in the earlier phase. In the company, peer review [16] is carried out after the design activity. In peer review, no explicit test case is prepared, and thus TI_{α} is not recorded.

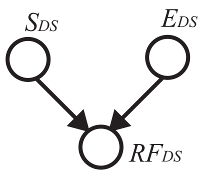


Figure 7. DAG for metric RF_{DS}

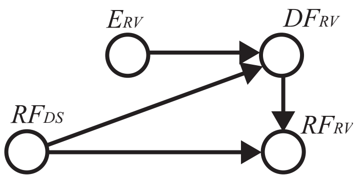


Figure 8. DAG for metric RF_{RV}

Obviously, the sum of the detected faults DF_{RV} and the residual faults RF_{RV} is equal to the total number of faults RF_{DS}. Thus we obtain the following relation among these metrics:

$$RF_{RV} = RF_{DS} - DF_{RV}$$

Clearly, RF_{RV} is depend on RF_{DS} and DF_{RV}.

Next, we consider on the DF_{RV} in more detail. In review activity, faults are discovered from the residual faults of the product. Moreover, the number of detected faults is affected by the review effort. Then, DF_{RV} is affected by RF_{DS} and E_{RV}.

Finally, we obtain DAG shown in Figure 8.

(3) DAG for SDB The activities in software test & debug (SDB) are also for removing the residual faults. Then, the structure of DAG for SDB is almost the same as that for RV. The difference between two DAGs is that the number of the test items TI_{SDB} is recorded in the software debug activity . Since metric TI_{SDB} is related to the coverage of the test, we consider that TI_{SDB} affects the number of detected faults DF_{SDB}. Furthermore, it is clear that TI_{SDB} affects the effort E_{DB}.

Finally, we obtain DAG shown in Figure 9.

(4) DAG for MDB The activities in hardware test & debug (MDB) is also for removing the residual faults. By applying the similar disscussions in DAG for SDB, we can get DAG shown in Figure 10 for MDB.

(5) DAG for AT The activities in acceptance test & debug (AT) also have the same properties as the software debug activities. Thus, we obtain DAG shown Figure 11.

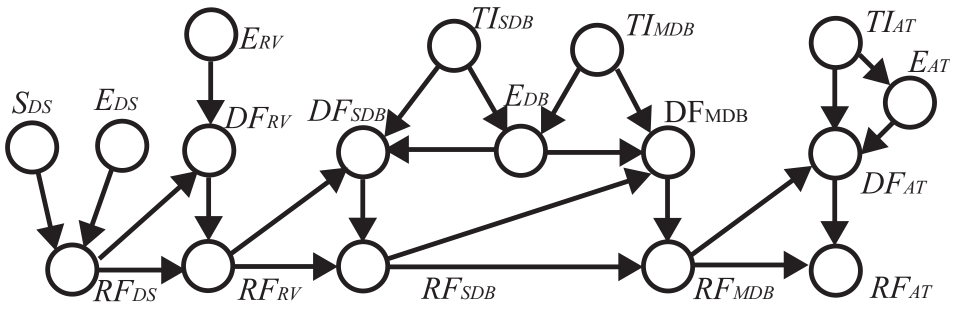


Figure 12. DAG for prediction model

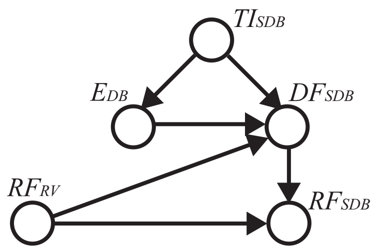


Figure 9. DAG for metric RF_{SDB}

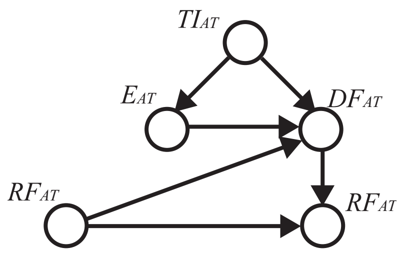


Figure 11. DAG for metric RF_{AT}

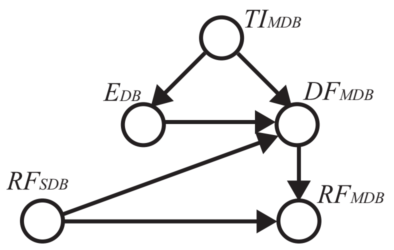


Figure 10. DAG for metric RF_{MDB}

Finally, by integrating these five DAGs shown in Figure 7, 8, 9, 10, and 11, we get a new DAG shown in Figure 12.

4.4 Assignment of Probability Distribution

Next, we assign probability distribution to each variable which exactly corresponds to a node in the integrated DAG.

However, there is a difficulty caused by smallness of the number of data of past projects or the volume of knowledge data, that are used to define probability distribution. In such case, we cannot use the value of the metric itself directly. Similarly, if the number of the project data that has a certain value is small, then we cannot assign such distributions for the variables that assures good prediction of final quality.

Then, we propose to group the values of metrics. However then we face the next difficulties: (1) how many groups we define and (2) how we decide the range of the values for each group.

For the first question, we decide the number of groups to be three. Then, we call three groups of RF_{AT} as “Good”, “Average”, and “Poor”, respectively. For the other metrics, we call three groups as “Small”, “Medium”, and “Large”, respectively.

For the second question, we have two straightforward methods. The one is to decide the range according to the value (simply called, “range by value”). The other is to decide the range according to the rank (simply called, “range by rank”) in descending order of values.

Now, we consider the range for the metric RF_{AT}. Since the model is used for prediction of the amount of residual faults, the value 0 should be a single group. The criterion for “Poor” project is determined by the company according to the expoerts’ opinion. Each dataset is collected from the same department of development and then required reliability is similar degree for all projects. Thus, for all projects, we can determine the criteria for “Poor”. Thus, we finally decide the range for RF_{AT} as shown in Table 6.

For the other metrics such as S_{DS}, E_{DS}, and so on, “range by value” is inadequate since the distributions of the values are biased. Thus, we adopt “range by rank” for these metrics. In this paper, in order to avoid subjective division, we

Table 6. The range of the values for RF_{AT}

Name of groups	Range of values
Good	0
Average	1
Poor	>1

Table 7. Sample Probability Distribution

RF_{AT}	probability(%)
Good	30%
Average	50%
Poor	20%

decide the range of the metrics such that each range contains the same number of data according to the rank.

4.5 Application Procedure

Here, we show an outlines of procedure which applies the BBN model to project data.

The procedure consists of two phases:

Phase 1: Construction of BBN Model

Step 1: Construction of DAG Based on abstract development process and metrics, construct a DAG (as described in subsection 4.3).

Step 2: Determination of Assignment Policy According to the expert's opinion and analysis result of the actual data, determine the assignment policy (as decribed in subsection 4.4).

Step 3: Assignment of Probability Distribution By using the dataset of the past projects, assign the probability distribution to the DAG. By this assignment, the prediction model is complete.

Phase 2: Calculation of Target Metric

Step 4: Classification of Nodes in DAG According to the requirement (that is, which metrics data can be collected and which metrics is wanted to know), classifty the nodes in the DAG into three types: ●, ○, and ◎. We must assign a concrete value for ●, but we don't give any value for ○. The ◎ represents a target metric (in our paper, RF_{AT} corresponds to ◎).

Step 5: Assignment of Value to Node ● We assign a value to each node ● in the DAG. The value itself is specified in the requirement.

Step 6: Evaluation of Probability for ◎ Based on BBN model constructed in Phase 1 and assigned values in Step 5, we calculate the probability for RF_{AT}. In order to perform such calculation, we use Netica, which is Bayesian network development software developed at Norsys Software Corp. (http://www.norsys.com). Then, a group which has the highest probability is determined.

For example, if the result of the probability distribution is as shown in Table 7, then we decide "Average" as the result of the prediction.

Example 1: Consider an example DAG in Figure 1 as the result of Step 1. Then, we may assign probability distributions shown in Tables 1, 2, 3, 4, and 5 at Step 3. Then at Step 4, we get a classification shown in Figure 3, and assign "T" to all nodes v_1, v_2, and v_3 at Step 5. Finally (as already mentioned in subsection 2.1), we get $P(v_5 = T) = 0.46$ at Step 6.

5 Model Evaluation

5.1 Outline of Evaluation Procedure

In order to evaluate the proposed model, we conducted two experimental evaluations (Preliminary and Specific evaluations) by applying the following two datasets (See Figure 14):

- **DATA1**
  This dataset is used, at Step 3 of both evaluations, to assign the probability distribution of the constructed model (See Figure 14(a)). Then it is used at Phase 2 of preliminary evaluation (See Figure 14(b)). The number of the data is 51.
- **DATA2**
  This dataset is not used at Phase 1, but used at Phase 2 of specific evaluation (See Figure 14(c)). The number of the data is 47.

By using these two datasets, we perform the following two evaluations as follows:

1. **Preliminary evaluation** (See Figure 14(a) and 14(b))
   At Step 1, for a given abstract development process (shown in Figure 5) and a set of metrics (explained in Section 4(B)), we construct a DAG shown in Figure 12. Then at Step 3, we calculate probability distribution for each metric using **DATA1**, and assign it to each node in the DAG. At the end of this step, CPTs are assigned

		RF_{DS} (%)		
S_{DS}	E_{DS}	Small	Medium	Large
Small	Small	90.7	9.0	0.3
Small	Medium	44.6	55.0	0.4
Small	Large	30.4	35.9	33.7
Medium	Small	31.9	67.7	0.4
Medium	Medium	0.4	58.2	41.4
Medium	Large	18.7	17.5	63.8
Large	Small	0.2	99.7	0.2
Large	Medium	0.4	76.1	23.5
Large	Large	11.5	11.6	76.9

Figure 13. CPT for RF_{DS}

to each node and probability distribution is obtained. For example, we show the CPT for RF_{DS} in Figure 13.

Next, at Step 4 we get a classification shown in Figure 15 (as Case Study 1). Then, we assign a value to ● in the DAG in Figure 15. Finally, we calculate probability for the target node ◎, and compare the result with **DATA1**. The analysis result is summerized in Table 8.

2. **Specific evaluation** (See Figure 14(a) and 14(c))
   In this evaluation, we use the same model as Preliminary evaluation. That is, the prediction model is constructed by using **DATA1**.

   Then we execute evaluation by using **DATA2**. At Step 4, we get a classification shown in Figure 16 (as Case Study 2). Then, we assign a value to each node ● in the DAG in Figure 16. Finally, we calculate the probability for the target node ◎, and compare the result with **DATA2**. The analysis result is summarized in Table 9.

In order to perform evaluation, we use the following three criteria:

- Error rate
  It represents the ratio of the incorrect prediction. By this criterion, we can calculate the accuracy of the prediction also. Then, the smaller the error rate is, the better the model is.

- Fisher's exact test
  This is the statistical test for verifying the correlation between two variables. By using this testing, we can verify whether there is the relationship between the result of prediction and the actual result (which is recorded as metric).

As mentioned, the purpose of this research is to detect risky projects that contain many faults though SRGM assures good quality of final products. Since all the projects in both **DATA1** and **DATA2** are assured by the SRGM, predicting such risky projects is important to conclude that the proposed model is applicable for the prediction. Then, we also use the following criterion.

- Prediction for "Poor" projects
  It evaluates the accuracy of the prediction for "Poor" projects only (rather than all projects).

5.2 Experimental Evaluation

Preliminary Evaluation The result of prediction is shown in Table 8. In this table, each row represents the actual value of the data. On the other hand, each column represents the predicted value of the model. For example, there exist 22 projects which are estimated to be "Poor". Among them, 13 projects were really "Poor". However, 5 projects are "Good", and 4 projects are "Average". Intuitively, the larger the number on the diagonal is, the more precise the prediction is.

From this table, we calculate the error rate of the prediction model (based on classification of nodes in Figure 15) as follows:

$$\begin{aligned}\text{error rate} &= \frac{51-(12+8+13)}{51}\times 100\\ &= 35.29(\%)\end{aligned}$$

Thus, the model seems to predict relatively well.

Next, we apply Fisher's exact test. Here, we assume the null hypothesis H_0 as follows:

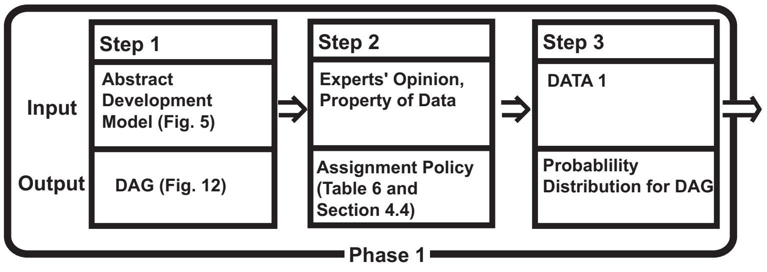
> "There is no correlation between the result of prediction and the actual result"

As the result of this test, p-value is $p = 0.0002$ and then the null hypothesis H_0 is rejected at significance level $\alpha = 0.05$. This implies that there is the correlation between the prediction and the actual development.

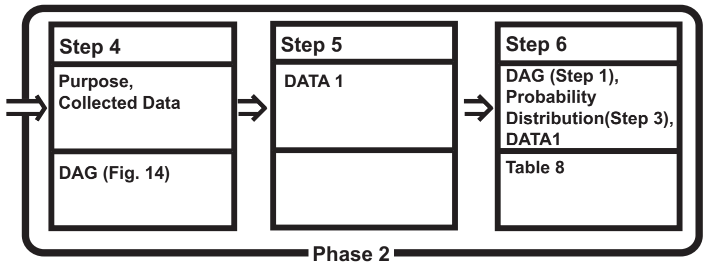
Finally we evaluate the accuracy of the prediction for "Poor" projects. As shown in the bottom row of Table 8, the total number of real "Poor" project is 17. On the other hand, the number of projects that are predicted to be "Poor" is 13. Then, the accuracy of the prediction for "Poor" project is calculated as follows:

$$\begin{aligned}\text{accuracy rate} &= \frac{13}{2+2+13}\times 100\\ &= 76.47(\%)\end{aligned}$$

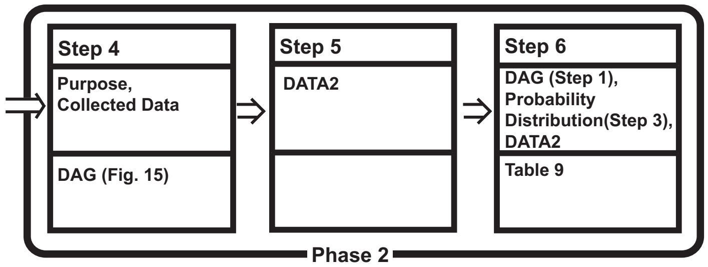
Observing Table 8 more precisely, we found that the constructed model predicts the amount of residual faults rather excessively. For example, the number of projects that are really "Good" but are predicted to be "Average" or "Poor" is larger than the number of projects that are really "Poor" but are predicted to be "Good" or "Average".


Input
Output
Step 1
Abstract Development Model (Fig. 5)
DAG (Fig. 12)
Step 2
Experts' Opinion, Property of Data
Assignment Policy (Table 6 and Section 4.4)
Step 3
DATA 1
Probablility Distribution for DAG
Phase 1

(a) Phase 1 in Preliminary and Specific evaluations



(b) Phase 2 in Preliminary evaluation



(c) Phase 2 in Specific evaluation

Figure 14. Outline of Evaluations

Table 8. The prediction result for Preliminary Evaluation

Predicted			Actual
Good	Average	Poor	
12	4	5	Good
1	8	4	Average
2	2	13	Poor

Table 9. The prediction result for Specific Evaluation

Predicted			Actual
Good	Average	Poor	
15	2	5	Good
5	5	5	Average
1	2	7	Poor

This property is rather desirable and welcome to the SEPG in the company. Since the purpose of the model prediction is to assure the final software quality, we should not miss the “Poor” projects as much as possible.

Specific Evaluation The result is shown in Table 9. Compared with the result of Table 8, the prediction accuracy seems not well. Intuitively speaking, we can understand this difference by comparing two DAGs in Figure 15 and Figure 16. That is, in Specific evaluation we try to predict the final

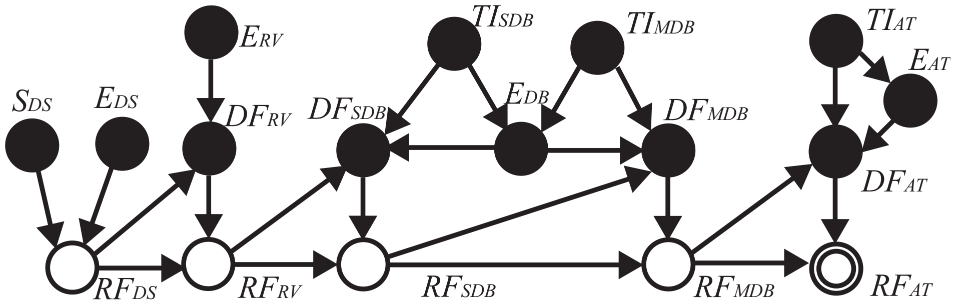


Figure 15. Classification of Nodes for Preliminary Evaluation

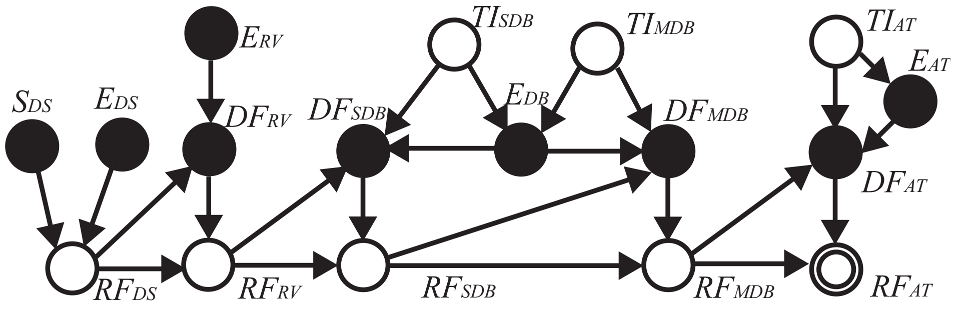


Figure 16. Classification of Nodes for Preliminary Evaluation

quality by using very few metrics data (that are assigned to node ● in Figure 16).

Actually, the error rate is $(47-(15+5+7))/47 = 20/47 =$ 42.55%. The error rate is increased by 7% compared with the error rate of Table 8. However, the p-value of Fisher's exact test is 0.01. Then, we can say that there is still the correlation between the result of the prediction and the actual result.

There seem to be two detailed reasons for increase of the error rate. The one is the difference of the departments, for which **DATA1** and **DATA2** are collected. As shown in Figure 14, Specific evaluation uses **DATA1** in model construction (Phase 1) and **DATA2** in model evaluation (Phase 2).

The other is the lack of TI_αs in evaluation (that is, in Figure 16, all TIs correspond to node ○). TIs count the number of test items, and thus they are considered to be essential to estimate test efficiency. Therefore, by excluding TIs in the classification shown in Figure 16, the accuracy of the prediction may be decreased.

However, as shown in the bottom row of Table 9, the accuracy of the prediction for "Poor" project is 70%. Then, this result implies that the proposed model works well for severe conditions.

Observing Table 9 in more detail, we found the similar trends as in Table 8. That is, in this case the proposed model also safely predicted in the sense that among 10 "Poor" projects, only 3 projects are not correctly estimated. This property is also acceptable to the SEPG in the company.

Considering these evaluation and observation, we can say that the model works well as the complement of SRGM. However, since one of purpose of the quality prediction is the reduction of the development cost, the accurate prediction is still important. This is one of the future works.

6 Conclusion

In this paper, we have proposed a new prediction model based on the BBN for final software quality. This prediction for final software quality is originated from the fact that the final quality of a few software products is not necessarily assured even if the SRGM is applied at the test.

The model construction based on the BBN consists of definition of DAG and assignment of probability distribution. In more detail, DAG contains the relationship between the metrics obtained from each activity in the software development process. Next, the probability distribution is assigned to the nodes in DAG which correspond to the metrics.

Then, by applying datasets that are collected in the actual project, we evaluate the proposed model with respect to three criteria: error rate, Fisher's exact test, and the accuracy for "Poor" projects.

As the results of these evaluations, we confirmed that the proposed model can be applicable to the prediction of final software quality.

In the future work, we must refine the model to improve the accuracy of the prediction. For this purpose, we must take the human factor into the model.

Additionally, the final goal of our research is the prediction in the early phase of the development process. Thus, we must investigate the property of the prediction with data obtained from the early phase, and refine the model according to the result of this investigation.

References

[1] D. B. Bisant and J. R. Lyle. A two-person inspection method to improve programming productivity. *IEEE Trans. on Software Engineering*, 15(10):1294–1304, 1989.

[2] E. Charniak and R. Goldman. A Bayesian model of plan recognition. *Artificial Intelligence*, 64:53–79, 1993.

[3] T. Compton and C. Withrow. Prediction and control of ada software defects. *Journal of Systems and Software*, 12:199–207, 1990.

[4] R. G. Cowell, A. P. Dawid, S. L. Lauritzen, and D. J. Spiegelhalter. *Probabilistic Networks and Expert Systems*. Springer-Verlag, 1999.

[5] M. E. Fagan. Advances in software inspections. *IEEE Trans. on Software Engineering*, 12(7):744–751, 1986.

[6] N. E. Fenton and M. Neil. Predicting software quality using bayesian belif networks. In *Proc. 21st Annual Software Engineering Workshop*, pages 217–230, 12 1996.

[7] N. E. Fenton and M. Neil. A critique of software defect prediction models. *IEEE Trans. on Software Engineering*, 25(5):675–689, 1999.

[8] N. E. Fenton and M. Neil. Software metrics and risk. In *Proc. 2nd European Software Measurement Conference(FESMA'99)*, pages 39–55, 1999.

[9] J. Forbes, T. Huang, K. Kanazawa, and S. Russel. The batmobile: Towards a Bayesian automated taxi. In *Proc. of the 14th International Joint Conference on Artificial Intelligence*, pages 1878–1885, 1993.

[10] A. L. Goel. Software reliability models: Assumptions, limitations, and applicability. *IEEE Trans. on Software Engineering*, pages 1411–1423, 12 1985.

[11] P. Haddawy. An overview of some recent developments in Bayesian problem solving techniques. *AI Magazine*, 20(2):11–20, 1999.

[12] M. H. Halstead. *Elements of Software Science*. Elsevier, 1977.

[13] E. Horvitz, J. Breese, D. Heckerman, D. Hovel, and K. Rommelse. The lumiere project: Bayesian user modeling for inferring the goals and needs of software users. In *Proc. of the 14th Conference on Uncertainty in Artificial Intelligence*, pages 256–265, 1998.

[14] International Standard Organization. *IEEE Standard Glossary of Software Engineering Terminology*. IEEE Std 610.12-1990, 1990.

[15] J. D. Musa, A. Iannino, and K. Okumoto. *Software reliability: measurement, prediction, application*. McGraw-Hill, 1987.

[16] M. C. Paulk, C. V. Weber, S. M. Garcia, M. B. Chrissis, and M. Bush. Key practice of the capability maturity model, version 1.1. Technical Report CMU/SEI-93-TR-025, Software Engineering Institute, 1993.

[17] I. Sommerville. *Software Engineering*. Addison-Wesley, MA, 4 edition, 1992.

[18] Y. Takagi, T. Tanaka, N. Niihara, K. Sakamoto, S. Kusumoto, and T. Kikuno. Analysis of review's effectiveness based on software metrics. In *Proc. of 5th International Symposium on Software Reliability Engineering*, pages 34–39, 1995.

[19] T. Tanaka, K. Sakamoto, S. Kusumoto, K. Matsumoto, and T. Kikuno. Improvement of software process by process description and benefit estimation. In *Proc. of 17th International Conference on Software Engineering*, pages 123–132, 1995.