PAPER Special Section on Software Engineering for Embedded Systems

Constructing a Bayesian Belief Network for Predicting Final Quality in Embedded System Development

Sousuke AMASAKI[†a)], **Yasunari TAKAGI**[††b)], ***Nonmembers***, **Osamu MIZUNO**[†c)], ***and*** **Tohru KIKUNO**[†d)], ***Members***

SUMMARY Recently, software development projects have been required to produce highly reliable systems within a short period and with low cost. In such situation, software quality prediction helps to confirm that software product satisfies required quality. In this paper, by using Bayesian belief network (BBN), we try to construct a prediction model based on relationships elicited from embedded software development process. According to a characteristic of embedded software development, we especially propose to classify test and debug activities into two distinct activities on software and hardware. Then we call the proposed model "the BBN for embedded software development process". On the other hand, we define "the BBN for general software development process" to be a model which doesn't consider such a classification, but merge them into a single activity. Finally, we conducted experimental evaluations by applying these two BBNs to actual project data. As the results of experiments, we proved that the BBN for embedded software development process is superior to the BBN for general development process and is applicable effectively for practical use.

key words: *Bayesian belief network, causal model, software quality prediction*

1. Introduction

Recently, embedded systems have been used for many types of systems including critical ones. An embedded system is used in many areas such as vending machine, cellar phone, home appliances, etc. Formerly, software used in an embedded system has been used mainly for controlling hardware. However, the purpose of embedded system has been diversified recently according to growth of demand for embedded system. In such situation, software in embedded system gets to play important role. For making embedded system more flexible, realizing functionality by software is prior to that by hardware. Thus, many functionality is realized by software instead of hardware.

Increase of demand for software causes increase of size and complexity of software. On the other hand, in terms of market strategy, duration for development is required to be shorten. That is, although recent embedded systems become complicated steadily, demands for cost, quality, and duration become severe. These problems are accelerated by the characteristics of embedded software [1]. An embedded software is required to achieve real-time processing with various hardware with severely constrained resources. Furthermore, recent embedded system deploys widely once it releases. This means that it is difficult to replace a system including faults.

In such situation, it is required for software development projects that highly reliable systems should be produced strictly within a short period and with low cost. Software quality prediction is valuable for the requirement because it helps to confirm that software product satisfies required quality. This is useful for avoiding wasteful effort and shipment of low quality product.

For general software development projects, there are extensive research for software quality prediction. Regression models with several metrics are often used as a method for predicting risky projects [2–4]. However, the regression models have the following problems [5].

First, metrics that are highly correlated cannot be used in the regression model simultaneously. Avoiding this difficulty is possible by introducing a new composite metric of these correlated metrics. However, we want to use values of metrics directly, since the values make it easy to investigate the relationship among metrics. Second, the regression model cannot be applied in a case such that the values of the explanatory variable in the model are unknown. That is, when we try to use the regression model, all values of the explanatory variable must be collected. The metrics that affect the final quality of software include the ones 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, applying the regression model is difficult.

Thus, we try to apply the Bayesian belief network (BBN) [6] as a modeling method to find risky project. The BBN is one of the methods for modeling systems that include causal relationships among variables. The BBN can handle uncertainties such 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 a decision or diagnose by using the probabilities for metrics whose value is unknown. This feature of the BBN is useful for solving the two problems of the regression model just mentioned. Furthermore, the BBN helps to model characteristics of embedded software development process differing from general software development process since it is constructed by using rela-

[†]Graduate School of Information Science and Technology, Osaka University, 1–5 Yamadaoka, Suita-shi, Osaka 565-0871, Japan
[††]Social Systems Company, OMRON Corporation, Japan
a) E-mail: amasaki@ist.osaka-u.ac.jp
b) E-mail: y-takagi@ist.osaka-u.ac.jp
c) E-mail: o-mizuno@ist.osaka-u.ac.jp
d) E-mail: kikuno@ist.osaka-u.ac.jp

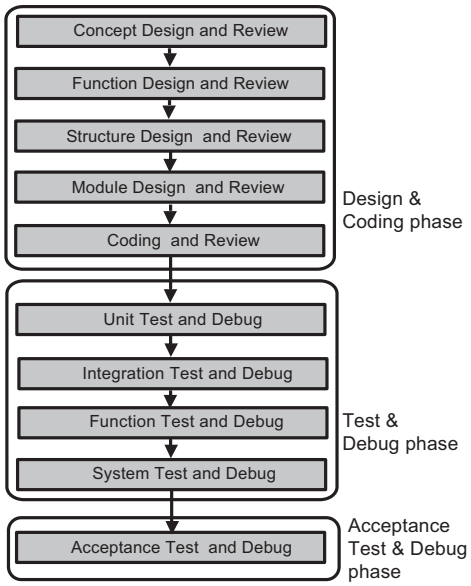


Fig. 1 Development process

tionships among diverse factors. This feature of the BBN is more suited than regression model when used for our purpose (that is to say, used for embedded software development).

In this paper, we propose a prediction model for final software quality of embedded software. We apply the BBN to embedded software development process. Thus, we also can handle uncertainties such as probabilistic events in the proposed model. Additionally, in order to deal with embedded software development process, we newly try to classify several activities in test and debug phase into two distinct activities on software and hardware. By the classification, we aim at realizing high prediction rate of the proposed model.

The rest of this paper is organized as follows: Section 2 explains target development process. Section 3 describes the flow of our study and mentions objectives. In Sect. 4 we construct abstract development process. We then construct the BBN in Sect. 5, and evaluate empirically the BBN in Sect. 6. Finally, Sect. 7 concludes this paper, and Appendix explains the BBN briefly.

2. Target Development Process

Figure 1 shows an overview of a development process in the company cooperating with us (provisionally named Company A). The development process is the overlapping ordinal waterfall model [7]. This process consists of three successive phases: the design and coding phase, the test and debug phase, and the acceptance test and debug phase.

The characteristics of the phases in this company are as follows:

(1) Design and Coding

This phase is divided into five activities: Concept design, Function design, Structure design, Module design, and Coding. One characteristic of design and coding phase in Company A is that review activity is introduced after each activity. A review activity not only improves the quality of an artifact but also helps software development organizations reduce their cost of producing software. [8]. In Company A, peer review [9] is carried out. The software engineering process group in the company establishes several guidelines for a review activity. One guideline directs at least 15% of the total effort for a design and coding phase to be assigned to a review activity [10]. After design activities, executable modules are coded.

(2) Test and Debug

This phase consists of four activities: Unit test and debug, Integration test and debug, Function test and debug, and System test and debug. In the company, the test and debug on software is carried out in the first two activities, and the test and debug on hardware is carried out in the last two activities. Test and debug on hardware is a feature of development process for embedded software.

(3) Acceptance Test and Debug

This phase (that is, this activity) is different from the test and debug on hardware in the test and debug phase. While a product is tested by developers themselves in the last two activities in the test and debug phase, it is tested by a software quality assurance team, which is organized apart from the development team.

3. Flow of Our Study

Figure 2 shows an outline of our study. Roughly speaking, we have the following two objectives in this study.

(1) For predicting final quality in embedded software development, we propose a new BBN with such a key feature that test and debug on hardware is dealt distinctly from test and debug on software. (Normally these two activities are summarized into a single test and debug.) Then we show that the proposed model can predict final quality quite well.

(2) In order to show the effectiveness of the key feature mentioned in (1), we also construct a BBN for general software development process for predicting final quality. (We imply that the BBN for general software development process has a single test and debug.) Then we show the superiority of the proposed BBN empirically to the BBN for general software development process.

The left hand side of Fig. 2 shows an outline of construction of the BBN with key feature for embedded software development process. The right hand side shows the construction of the BBN for general software development process.

In the following sections, we call the BBN with the key feature as “embedded model”. On the other hand, we call the BBN with single test and debug as “general model”.

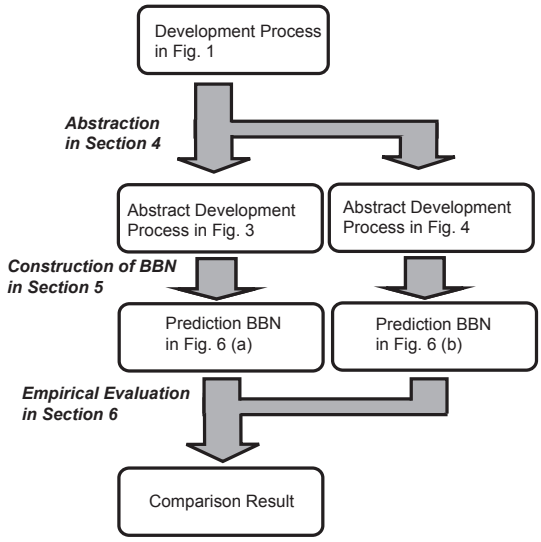


Fig. 2 Flow of Our Study

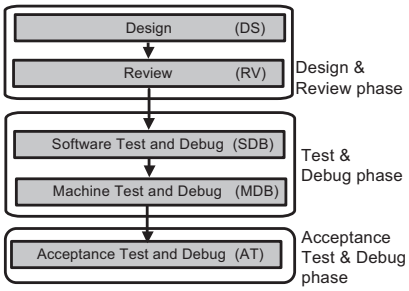


Fig. 3 Abstract development process for embedded model

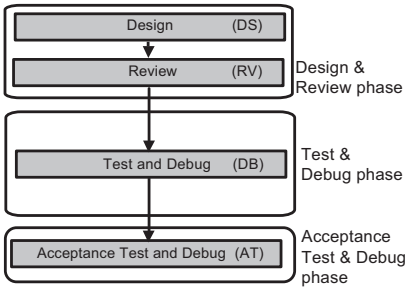


Fig. 4 Abstract development process for general model

4. Abstract Development Process

As the first step, we construct abstract development process from actual development process. Here, the actual development process is given in Fig. 1.

Now we explain how to construct abstract development process for embedded model shown in Fig. 3 Later we describe construction of the one for general model shown in Fig. 4.

(1) Firstly for design and coding phase, we construct a pair of design activity and review activity, as shown in Fig. 3. That is, from four kinds of design and review in Fig. 1, we define two activities: Design Activity and Review Activity as shown in Fig. 3.

(2) Next for test and debug phase, we propose a new trial based on the fact that the first two test and debug in Fig. 1 are executed for software and the last two test and debug are for hardware. Then, we define two activities: Software Test and Debug and Machine Test and Debug as shown in Fig. 3.

(3) Finally for acceptance test and debug phase, we define a single activity as shown in Fig. 3.

On the other hand, abstract development process for general model is constructed by the similar steps mentioned above. The difference is only for test and debug phase. According to general view point, four test and debug activities in Fig. 1 are merged into a single test and debug. As the result, abstract development process for general model shown in Fig. 4 is obtained.

5. BBNs for Software Quality Prediction

Then, we construct a prediction model by using the BBN from abstract development process. Detailed explanation about the BBN is described in Appendix. In order to construct the BBN, we need to construct DAG and CPT. However, method for CPT is already established and can be carried out automatically. Thus, in the following subsections, we describe only how to construct the DAG for prediction model. The resultant DAGs are shown in Fig. 5 and 6.

5.1 Approach to Model Construction

Based on the abstract development process in Sect. 4, we construct the prediction model for final software quality using the BBN. Because the concept of software quality consists of diverse facets of software, we should consider various criteria, which are usually assessed by collectable metrics. In such metrics, the amount of residual faults is considered one of the most important factors because it represents the amount of inconsistency between the specification and the implemented product. Furthermore, the development members in the company regard this metric as the most important quality measurement. Thus, in this paper, we deal with the amount of residual faults in a product as a factor of software quality. Concretely, our proposed models predict whether the amount of residual faults in a product is acceptable at the end of the acceptance test and debug activity.

The following approach represents the policy of model construction:

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

(2) Calculate the amount of residual faults in each activity from the metrics recorded in the corresponding activity.

5.2 Software Metrics

In order to represent relationships between development process and final software quality, we use software metrics obtained in each activity. The metrics used in the proposed models are classified into five groups: S_α, E_α, DF_α, TI_α, and RF_α where α denotes an activity in Fig. 3 and 4. For example, S_{DS} denotes product size in DS (design activity), and E_{DS} denotes effort in DS. The detailed definitions are given as follows:

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

All metrics without RF_α can be collected in an actual software development project. Since each activity in the abstract development processes consists of some activities, the values of metrics E_α, DF_α, and TI_α are given by summing up the values of the corresponding activities (shown in Fig. 1). For example, effort E_{DS} is calculated as the total sum of effort at the concept design, function design, structure design, module design, and at coding. Here, Effort metrics for software test and debug activitiy and machine test and debug activity are not defined since these metrics are not recorded separately. Thus, we use E_{DB} to DAGs for SDB and MDB.

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. In order to overcome this deficiency, we assume that RF_{AT} is equal to the number of detected faults during the six months after shipping. Thus, we can define the total number of faults RF_{DS} (that is, the residual faults after 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 are calculated by subtracting the detected faults DF_α in each activity α from the residual faults in the previous activity.

When we try to predict software quality, human factors such as ability and expertise, should be considered key metrics for the prediction. However, these metrics are not recorded, and thus we do not consider them in this paper. Considering such human factors, however, still remains an important task for future work.

In order to use these software metrics for the BBN, we next discretize them. First, we discretize all software metrics without RF_{AT}. The number of classes for discretized software metrics is decided as three in this paper. For example, these classes are called “Large”, “Medium”, and “Small”, respectively. Next, thresholds for each class in discretized software metrics are decided. In this paper, for each metric, these are decided by using actual dataset such that sorted dataset of a metric are divided equally. Next, we decide a threshold for the metric RF_{AT}. Since the model is used for software quality prediction, zero should be a single group. Thus, classes for RF_{AT} are “Good”, which means $RF_{AT} = 0$, and “Poor”, which means $RF_{AT} > 0$, respectively.

5.3 Construction of DAG

In this subsection, we define prediction model using the BBN for abstract development processes shown in Fig. 3 and 4. First, we construct DAGs for each activity. For embedded model, we construct DAGs for five activities based on relationships among metrics related to the corresponding activities. Then, we integrate these DAGs into a DAG that represents the whole development process.

Figure 5 shows resultant DAGs for each activity. Please note that DAGs for design activity, review activity, and acceptance test and debug activity are common to both abstract development processes for embedded model and general model. DAGs for software test and debug activity and machine test and debug activity are shown in the left side of Fig. 5 and DAGs for test and debug activity is shown in the right side of Fig. 5. In the following, we mainly explain in detail how to construct DAGs for embedded model. For general model, we explain only about a DAG for test and debug activity.

(1) DAG for Design Activity (common)

Here, we assume that faults are introduced by design activity, and that residual faults are removed by other activities such as review, test and debug in Fig. 3 and 4. We assume that metrics S_{DS} and E_{DS} in design activity (DS) affect the number of the introduced faults 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}.

(2) DAG for Review Activity (common)

Review is carried out in order to remove faults remaining or introduced in the earlier phase. In the company, peer review [11] is carried out after the design activity. In peer review, no explicit test case is prepared, and thus TI_α is not recorded.

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} depends on RF_{DS} and DF_{RV}.

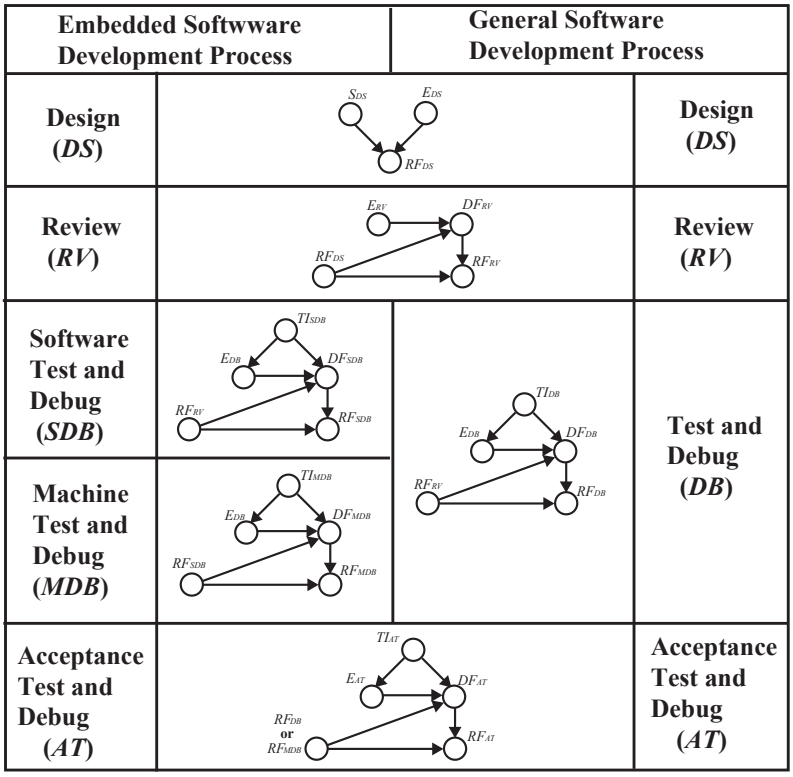


Fig. 5 DAGs for development processes

Next, we consider 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}.

(3) DAG for Software Test and Debug Activity

The activities in software test & debug (SDB) are also used to remove 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, TI_{SDB} clearly affects the effort E_{DB}.

(4) DAG for Machine Test and Debug Activity

The activities in the machine test & debug (MDB) are also for removing the residual faults. By applying similar discussions in DAG for SDB, we get the DAG.

(5) DAG for Test and Debug Activity

In general development process, test and debug (DB) is performed only for software. Thus, activities in test and debug phase are integrated into a single test and debug activity, as shown in Fig. 4. As same as integration of activities in software test and debug activity, we simply add TI_{SDB} and TI_{MDB}. This metric is defined as TI_{DB} in Subsection 5.2. Similarly, RF_{DB} and DF_{DB} are defined. Since the relationships between these metrics are the same ones, the DAG for test and debug activity is constructed as shown in Fig. 5.

(6) DAG for Acceptance Test and Debug Activity (common)

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

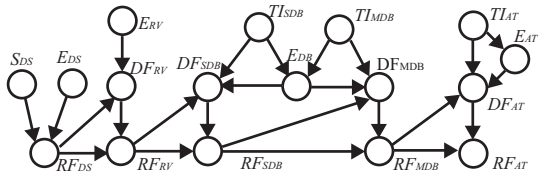
5.4 DAG for Prediction Model

Finally, DAGs in Fig. 5 are integrated into a prediction model. Figure 6 (a) and (b) show DAGs for embedded model and for general model, respectively.

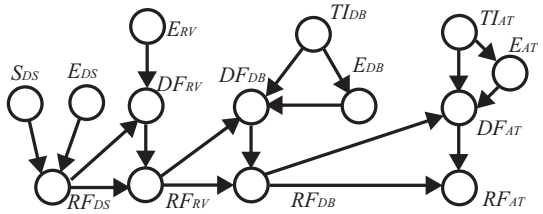
6. Evaluation of BBNs

6.1 Dataset

For empirical evaluation, we use actual project data obtained from Company A, which cooperates with us. The projects targeted in this paper are the development of computer control systems with embedded software in Company A. 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 developed software is delivered in



(a) Embedded Model



(b) General Model

Fig. 6 DAGs for prediction BBNs

the form of LSI chips, modification of faults after delivery is very expensive. Thus, high quality is especially required for the embedded software.

The dataset used in this paper consists of the actual project data of 51 projects, which have already finished their development. Each project started its development from 1995 to 1998.

6.2 Application Procedure

In order to evaluate the BBNs according to objectives mentioned in Sect. 3, we perform two experiments using the dataset.

- Self-application
  As a preliminary experiment, we first apply the dataset to a prediction model for both training and prediction. From the result of this experiment, we can see how well the model represents a relationship between a feature of software development process and final software quality.

- 10-fold cross validation
  The evaluation through k-fold cross validation method is one of the most common technique for model evaluation. The dataset is here split into k equally sized subsets. Then in i-th iteration ($i = 1, \ldots, k$), the i-th subset is used for testing the BBN that has been learned from all other remaining subsets. Notice that each instance of data is classified exactly once. The number of subsets k is usually set to 10. From this experiments, we can see how applicable the model is for practical use.

The results of predictions are evaluated using the following two perspectives:

Table 1 Self-application result for embedded model

Embedded		Predicted	
		Good	Poor
Actual	Good	11	10
	Poor	3	27

Accuracy rate: 75.51%
Fisher's exact test: p<0.01

Table 2 Self-application result for general model

General		Predicted	
		Good	Poor
Actual	Good	8	11
	Poor	7	23

Accuracy rate: 72.55%
Fisher's exact test: p<0.01

Table 3 10-fold cross-validation result for embedded model

Embedded		Predicted	
		Good	Poor
Actual	Good	10	11
	Poor	6	24

Accuracy rate: 66.67%
Fisher's exact test: p=0.06

Table 4 10-fold cross-validation result for general model

Embedded		Predicted	
		Good	Poor
Actual	Good	8	13
	Poor	7	23

Accuracy rate: 60.78%
Fisher's exact test: p=0.35

- Accuracy rate
  Accuracy rate represents the ratio of the correct prediction. This criterion implies the accuracy of the prediction. The larger the accuracy rate is, the better the model is. For evaluating both BBNs, we use this criterion.

- Fisher's exact test
  Fisher's exact test is the statistical test for verifying the correlation between two variables. By using this testing, we can verify whether or not there is a relationship between the result of a prediction and the actual result (which is recorded as a metric). This criterion is used for evaluating goodness of a prediction model. In this paper, we adopt significance level $\alpha = 0.1$.

6.3 Comparison

The prediction result of self-application is shown in Table 1. From this result, we can see that accuracy rate is 75.51%. It is relatively good accuracy rate. P-value of Fisher's exact test results in lower than 0.01. This implies that there is a correlation between actual software quality and predicted one. From these two, we can say that embedded model can represent a feature of development process for embedded

software.

Next, the prediction result of 10-fold cross-validation is shown in Table 3. This table says that accuracy rate is 66.67%. It is also relatively well though lower than the one of self-application. In the fact, p-value of Fisher's exact test is 0.06. This is still lower than significance level $\alpha = 0.1$. Thus, we can say that embedded model is applicable for practical use.

In contrast to embedded model, the prediction results of general model is not so good. For self-application, Table 2 shows that accuracy rate is 72.55%. This is slightly inferior to that of embedded model. However, p-value of Fisher's exact test is lower than 0.01. Thus, we can say that general model represent a feature of development process for embedded software. On the other hand, for 10-fold cross-validation, accuracy rate of Table 4 is 60.78%. P-value of Fisher's exact test is 0.35. This exceeds the significance level. Thus, we cannot say that there is a correlation between actual software quality and predicted one.

These results imply that distinction of test and debug activities on software and on hardware leads higher prediction rate and that prediction model based on embedded software development process is applicable for practical use.

7. Conclusion

In this paper, we proposed the prediction model for final software quality by using the Bayesian belief network. This model is based on embedded software development process.

In order to deal with embedded software development process, we especially classify several test and debug activities into two activities on software and on hardware. By applying the classification, we defined the BBN for embedded software development process. We also define the BBN for general software development process, in which such a classification is not applied.

Then, we evaluated both BBNs with empirical data set collected from actual projects developing embedded systems. As a result, we showed that the BBN for embedded software development process gives higher accuracy rate rather than the BBN for general software development process and is applicable for practical use.

In future work, refinement of the model to improve the accuracy of the prediction is needed. For this purpose, the human factor should be in the model. Additionally, the prediction in the early phase of the development process is also a challenging topics.

References

[1] H. Takada, "The recent status and future trends of embedded system development technology," IPSJ Journal, 2001.
[2] J.C. Munson and T.M. Khoshgoftaar, "The detection of fault-prone programs," IEEE Trans. on Software Engineering, vol.18, no.5, pp.423–433, 1992.
[3] J.A. Morgan and G.J. Knafl, "Residual fault density prediction using regression methods," Proc. 7th International Symposium on Software Reliability Engineering, pp.87–92, 1996.
[4] Y. Yokoyama and M. Kodaira, "Software cost and quality analysis by statistical approaches," Proc. 20th International Conference on Software Engineering, pp.465–467, 1998.
[5] N.E. Fenton and M. Neil, "A critique of software defect prediction models," IEEE Trans. on Software Engineering, vol.25, no.5, pp.675–689, 1999.
[6] R.G. Cowell, A.P. Dawid, S.L. Lauritzen, and D.J. Spiegelhalter, Probabilistic Networks and Expert Systems, Springer-Verlag, 1999.
[7] W.S. Humphrey, A Discipline for Software Engineering, Addison-Wesley, MA, 1995.
[8] M.E. Fagan, "Advances in software inspections," IEEE Trans. on Software Engineering, vol.12, no.7, pp.744–751, 1986.
[9] D.B. Bisant and J.R. Lyle, "A two-person inspection method to improve programming productivity," IEEE Trans. on Software Engineering, vol.15, no.10, pp.1294–1304, 1989.
[10] Y. Takagi, T. Tanaka, N. Niihara, K. Sakamoto, S. Kusumoto, and T. Kikuno, "Analysis of review's effectiveness based on software metrics," Proc. of 5th International Symposium on Software Reliability Engineering, pp.34–39, 1995.
[11] 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," Tech. Rep. CMU/SEI-93-TR-025, Software Engineering Institute, 1993.
[12] E. Charniak and R. Goldman, "A Bayesian model of plan recognition," Artificial Intelligence, vol.64, pp.53–79, 1993.
[13] J. Forbes, T. Huang, K. Kanazawa, and S. Russel, "The batmobile: Towards a Bayesian automated taxi," Proc. of the 14th International Joint Conference on Artificial Intelligence, pp.1878–1885, 1993.
[14] P. Haddawy, "An overview of some recent developments in Bayesian problem solving techniques," AI Magazine, vol.20, no.2, pp.11–20, 1999.
[15] 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," Proc. of the 14th Conference on Uncertainty in Artificial Intelligence, pp.256–265, 1998.

Appendix: Bayesian Belief Network

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 widespread areas [12–15] such as artificial intelligence, medical diagnosis, trouble shooting diagnosis, and decision making.

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 [6]. Formally, this graph is in the form of DAG (directed acyclic graph).

An example of DAG is shown in Fig. A·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 A·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 a dependent variable.

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

Table A·1 is an example of the CPT assigned to variable v_4, which shows the relationship among v_2, v_3, and v_4. Here, we assume that all variables are binary which take val-

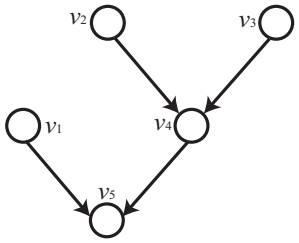


Fig. A· 1 Example of DAG

Table A· 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

ues "T" or "F".

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

Case 1: Values of all variables are known.
Case 2: Values of some variables are unknown.

In this paper, by utilizing a property in Case 2, we construct a prediction model.

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 A· 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. Therefore, $p(v_5 = T|v_1 = T, v_2 = T, v_3 = T, v_4 = T) = p(v_5 = T|v_1 = T, v_4 = T) = 0.7$ is obtained from Table A· 2.

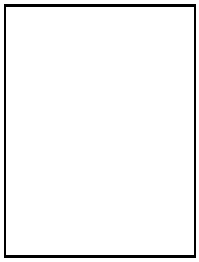
Next, we calculate the probability of $v_5 = T$ under the almost same condition. The difference is that a value of v_4 is uncertain. In order to calculate the probability of v_5, we need the value of v_4. However, in this case, the value of v_4 is unknown.

First, from Table A· 1, we 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 A· 2. Although v_4 is uncertain, we can calculate the probability of $v_5 = T$ by using Table A· 2 as follows:

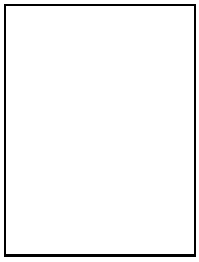
$$
\begin{aligned}
&p(v_5 = T|v_1 = T, v_2 = T, v_3 = T) \\
&= p(v_5 = T|v_1 = T, v_4 = T) \times p(v_4 = T|v_2 = T, v_3 = T) \\
&\quad + p(v_5 = T|v_1 = T, v_4 = F) \times p(v_4 = F|v_2 = T, v_3 = T) \\
&= 0.7 \times 0.4 + 0.3 \times 0.6 \\
&= 0.46
\end{aligned}
$$

Table A· 2 Conditional probability table for v_5

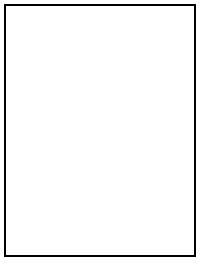
		v_5	
v_1	v_4	T	F
T	T	0.7	0.3
	F	0.3	0.7
F	T	0.8	0.2
	F	0.3	0.7



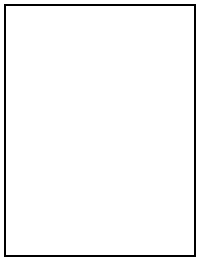
Sousuke Amasaki received B.E. degree in computer science and system engineering from Okayama Prefectural University in 2000 and M.E. degree from Osaka University in 2003. He is currently a Ph.D course student in Graduate School of Information Science and Technology, Osaka University. His current research interests include quantitative evaluation of software process and software quality prediction.



Yasunari Takagi received B.E. degree in information and computer science from Nagoya Institute of Technology in 1985. He has been working for OMRON Corporation. He is currently a Ph.D course student in Graduate School of Information Science and Technology, Osaka University. His current research interests include software process improvement activities in industries and software quality assurance of embedded software systems.



Osamu Mizuno received M.E. and Ph.D. degrees from Osaka University in 1998 and 2001, respectively. He is currently an Assistant Professor in Graduate School of Information Science and Technology at Osaka University. His research interests include methodologies of software process improvement and risk evaluation and prediction in software development. He is a member of the IEEE.



Tohru Kikuno received M.Sc. and Ph.D. degrees from Osaka University in 1972 and 1975, respectively. He joined Hiroshima University from 1975 to 1987. Since 1990, he has been a Professor of the Department of Information and Computer Sciences at Osaka University. Since 2002, he has been a Professor of Graduate School of Information Science and Technology at Osaka University. He also holds a Director of Osaka University Nakanoshima Center from 2004.
His research interests include the analysis and design of fault-tolerant systems, the quantitative evaluation of software development processes, and the design of procedures for testing communication protocols.
He is a senior member of IEEE, a member of ACM, IEICE (the Institute of Electronics, Information and Communication Engineers), and a fellow of IPSJ (Information Processing Society of Japan). He received the Paper Award from IEICE in 1993.