

Identifying Metrics for Commercial-Off-the-Shelf Software with Inductive Inference Based on Characteristic Vectors*

CHONGWON LEE, BYUNGJEONG LEE, JAEWON OH[†] AND CHISU WU^{††}

School of Computer Science

University of Seoul

Seoul 130-743, Korea

[†]*School of Computer Science and Information Engineering*

The Catholic University of Korea

Seoul 137-701, Korea

^{††}*School of Computer Science and Engineering*

Seoul National University

Seoul 151-742, Korea

Nowadays, many users and organizations are interested in acquiring COTS (commercial-off-the-shelf) software products instead of building software systems themselves as acquisition reduces development costs. COTS products are usually provided in a packaged style without the source code but with many ready-to-use functions. To assure the proper level of quality, many organizations provide quality evaluation and certification services for COTS. Generally, their vendors are reluctant to disclose the source code. Thus, the major way of quality evaluation and certification requires dynamic behavior testing, essentially black-box testing. Since observing every aspect of external software behavior is almost impossible, it is crucial to designate an adequate range for quality evaluation such as an adequate number of quality checklists or product quality metrics for external behavior testing. Hence, to establish rules of selecting quality evaluation criteria in systematic ways, there have been attempts to analyze and utilize the past records of software evaluation based on artificial intelligence techniques. A Bayesian belief network (BBN) is one of the methods using an inductive inference based on prior experiences. In this paper, we represent software as characteristic vectors having dependency relationships with the external product quality metrics. BBN is then used to infer the metrics for new software products.

Keywords: COTS software, characteristic vector, metric, inductive inference, black-box testing

1. INTRODUCTION

Nowadays, many developers and users have interest in the purchase of COTS software products rather than building the software products themselves. According to a report of the IDC¹, 63% of the market share for global software licenses is occupied by the 100 global enterprises which are dominant in the field of COTS software [1]. COTS products are software systems that can be used without modification [2]. Thus, only external behavioral characteristics can be measured to certify the quality of COTS as source

Received May 21, 2007; revised August 20, 2007; accepted October 25, 2007.

Communicated by Sy-Yen Kuo.

* This study was supported by the Research Fund, 2007 of The Catholic University of Korea and supported by the Korea Science and Engineering Foundation (KOSEF) grant funded by the Korea government (MEST) (No. R01-2006-000-11150-0). [†] Corresponding author.

¹ <http://idc.com>.

code is usually hidden and not disclosed to the public. When such quality measurements are carried out, diverse external behavior checklists or external product quality metrics are examined and chosen to set up the criteria for measurement. Software systems are getting more complex. For example, the most extensive software systems are capable of handling 10^{20} different situations [3]. Therefore, it is considered to be very difficult to achieve high reliability [4]. Considering such difficulty, the main objective of software quality evaluation is merely to determine that an approximate quality level with respect to the characteristics and operating environments of software products is assured rather than pursuing perfection [5].

Regarding such a situation of no perfection reasonably possible in software quality evaluation, there is an analytical opinion that quality certification in computing should follow the embedded software model making the end-users aware of application domains for which the software products have specialized [6]. Thus, it is important to decide adequate criteria of quality measurement, external product quality metrics, to describe the surrounding environment and characteristics of the software products being certified. If the exact specification of product quality regarding the certified products of COTS software can be provided for the end-users, the end-users may estimate the expected reliability of certified products for their specialized application domains. Generally, a quality evaluation for COTS is carried out in black-box style since there is no source code. Therefore, the external software quality metrics specified in ISO/IEC 9126-2 [7] are often referenced to create the evaluation rule sets for testing COTS.

At this time, there has not been any international standard for selecting external product quality metrics that are tailored for specific software domains. However, certifying software systems composed of many different COTS components requires customized evaluation methods [8]. One of the common approaches is adopting the methods of deductive inference. These methods are based mainly on the knowledge of experts or formal theory. For example, by analyzing requirements or referencing a formal framework, a quality evaluator can designate the test coverage and test cases for black-box testing. Another useful approach is adopting inductive inference which is the theory of prediction based on observations such as predicting the next symbol based upon a given series of symbols. By analyzing individual cases for quality evaluation, specific relationships between the software characteristics and product quality metrics can be identified. These newly identified relationships can be used to infer appropriate metrics for a new COTS software.

So far, statistical methods have often been used to infer such relationships in the form of inductive inference. However, it has been noted that the statistical methods fail to describe casual relationships among the software characteristics and metrics [9]. Thus, alternative solutions, such as data mining or artificial intelligent methodologies, have been used to infer these vague relationships. Similar to data mining approaches, the certification meta-model has also been proposed to represent the relationships among the metrics and software groups although the expressed relationship has some limitations [10, 11].

In this paper, we adopt the Bayesian Belief Network (BBN) [12]. BBN is usually used to represent causal knowledge such as the fact that lightning causes thunder. When inferring specific relationships among the software characteristics and metrics, the dependency relationship can be regarded as a kind of causal knowledge style relationship.

In this paper, each COTS product represents itself using characteristic vectors. The characteristic vectors can be defined as some kind of fingerprint for the product. Just like the training data of neural networks, past software quality evaluation data is used to construct the BBN having a directed relationship of influence among characteristic vectors, software products and metrics. The characteristic vectors influence their corresponding software. Likewise, the software products influence their corresponding metrics according to the evaluation data. After the construction of the network, software certifiers can refer to joint post probabilities showing the appropriateness of metrics when evaluating a new COTS. This can be seen as a kind of computer-aided COTS software evaluation guideline.

This paper is organized as follows. In section 2, we describe existing software metric inference methods, the certification meta-model, and BBN. In section 3, we define the characteristic vectors for classifying COTS software. In section 4, we show a simplified example of metrics inference using a BBN. In section 5, we describe the reasons using Bayesian Belief Network and comparisons to other approaches. In section 6, we present an extended case study for an actual operation. The conclusions follow in section 7.

2. RELATED WORK

In this section, existing software metric inference methods and the certification meta-model are described. The concept of BBN is also briefly described.

2.1 Software Metrics Inference

Many approaches have tried to analyze past records of quality evaluation in the style of inductive inference rather than depending on the knowledge of experts or formal theory [9, 13, 14]. Using an approach based upon a genetic algorithm, selecting metrics are used to quantify the characteristics of source code written in an object-oriented language [13]. After acquiring the data for metric and quality drawn from existing software objects, diverse groups of software metrics are decided referring to the quality ranks assigned by experts using an approach involving genetic algorithms. However, there is no clear and distinctive relationship among the metrics and software objects.

Another method has been proposed to infer metrics using an interconnected multi-layer perceptrons grid [14]. The metrics are set up to quantify the characteristics of source code for software quality analysis. As a basic metrics inference mechanism, the method utilizes the self-learning capability of perceptrons. Through trial and error, the grid of perceptrons can classify the characteristics of source code for proper metrics. After this process, the grid of perceptrons requires only a small fraction of the input data to get output data used for producing consistent classification results. However, we should not overlook the possibility of biased output data occurring resulting from over-fitting on old training data.

Another technique is unsupervised learning [9]. Unsupervised learning searches for every possible combination of solutions without fixed guidance. The main idea is to increase the probability of finding optimal solutions. This kind of technique resolves the problem of over-fitting on old training data. This technique applies fuzzy clustering to analyze the existing software metrics database. The existing metrics are collected and

analyzed in the software development phases to control the quality of final software products. There are three data sets of software metrics forming fuzzy clusters. Each cluster also has software modules corresponding to its own specific characteristics and metrics. The clusters are ranked by the number of faults caused by their software modules. The modules showing a higher rate of faults are thoroughly managed to fix the faults. This technique does not rely on dichotomous reasoning. It allows for gradual improvement of the modules to be traced continuously. However, there is also the possibility of skewed reasoning resulting from the lack of background data.

The BBN is also used to infer the metrics or potential faults of software [15-17]. The limitation of a statistical approach is pointed out again while emphasizing the robustness of a BBN which can make decisions under vague conditions [15, 16]. Every factor that can be considered during software development, such as maturity of programmers and complexity of problems, is set to have dependency relationships among the factors on a BBN in order to express appropriateness for diverse situations in probability. However, there seem to be some kind of guideline needed for processing a large amount of prior cases data since many factors are not classified for proper domains.

The other case of the BBN application for metrics selection is estimating risk factors for the development of critical systems [17]. According to the statistical regression model, a software module showing a high rate of faults during tests is generally expected to cause problems when integrating with other modules. However, many actual cases report just the opposite. The alternative prediction model of module faults tries to express relationships among final software products and process-related factors regarding the entire software model lifecycle on the BBN. However, the alternative model does not mention how to specialize each different module.

Similar to selecting software metrics, many prior approaches adopted and utilized the knowledge of experts or formal theory in the style of deductive inference to select test cases [18, 19]. For black-box testing, which is commonly used for COTS software, this kind of inference is suitable when creating test cases from requirements that adequately exercise the behavior of a software system without regard to the internal structure such as source code. However, utilizing the formal theory or experts is not so easy for end-users.

2.2 The Certification Meta-Model

The certification meta-model is the model referenced by accrediting institutions to decide the certification model and programs [10]. Before using the meta-model, each metric should be assigned to the proper group on a hierarchical graph structure. When actually certifying, the meta-model is converted into a certification model. The certification model is then used to generate certification programs applied for actual evaluations of COTS software. The noticeable feature of the meta-model is the group-based approach with specialization [10] which is applied on the hierarchical graph structure. The approach tries to adopt an inductive methodology in order to extend the classification structure of certification gradually by referring to the past cases of certification. In other words, by using the data of past cases, redundant programs of certification could be avoided. Based on such infrastructure, the appropriateness of each metric could also be represented according to the type of package software products.

However, there are drawbacks in the meta-model. The hierarchical structure of classification must be constructed manually for package software products. Furthermore, there is no integrated approach to classification automation. Since manual classification depends on human evaluators, inconsistent and awkward metrics allocation can occur. Thus, a mechanism of metrics allocation which is based on an autonomous and general principle is necessary in order to avoid any controversy in connection with inconsistent criteria for the quality evaluation.

To complement the mechanism of metrics allocation, the Formal Concept Analysis has been adopted to automate the allocation of metrics [11]. However, Formal Concept Analysis could not represent the appropriateness of each metric in fine detail since there has been no support for representation of probability.

2.3 The Bayesian Belief Network

A Bayesian belief network is a probabilistic graphical model that represents a set of variables and their probabilistic independencies. For example, a BBN can be used to calculate the probability of a patient having a specific disease, given the absence or presence of certain symptoms, if the probabilistic independencies between symptoms and disease as encoded by the graph hold.

The fundamental notion of a BBN is based on Bayesian learning. When inferring with Bayesian learning, the outcome of the inference reacts to sensitively of environmental changes since Bayesian learning incorporates the past experiences data and the decisions of the operator for analysis [12]. Bayes theorem consists of the basic algorithm of Bayesian learning:

$$P(h|D) = \frac{P(D|h)P(h)}{P(D)}. \quad (1)$$

In this theorem, $P(h)$ is the prior probability of hypothesis h while $P(D)$ denotes the prior probability that training data D will be observed. $P(D|h)$ can be regarded as the probability of observing data D where h holds. $P(h|D)$ is the posterior probability of h .

When there are many variables to consider for a probability distribution, BBN is commonly used instead of the Bayes theorem. A BBN's primarily objective is to describe a joint probability distribution from a set of variables used to estimate the posterior probabilities. A node of a BBN, which corresponds to a variable, forms a dependency network with other nodes [20]. The algorithm which calculates the joint probability distribution of BBN can be expressed as:

$$P(y_1, \dots, y_n) = \prod_{i=1}^n P(y_i | \text{Parents}(Y_i)). \quad (2)$$

$\text{Parent}(Y_i)$ means the set of Y_i 's immediate parent nodes. Thus, $P(y_i | \text{Parent}(Y_i))$ means the conditional probability, which is related to the node Y_i , considering its parent nodes. In an example in Fig. 1, there are two parent nodes of animal characteristics which could cause "Monkey" to be true: either "fourLegged" is true or "climbingTree" is true. Except for the two parent nodes having a direct effect on the child node, "Monkey," "Monkey"

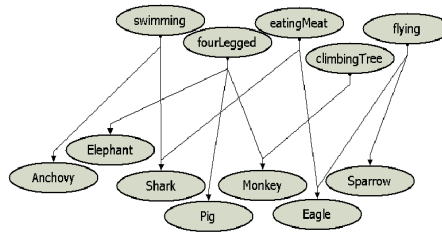


Fig. 1. Dependency relationship of variables.

itself is conditionally independent, or not under the influence of other parent nodes. The possible states of conditional probabilities of “Monkey” can be described like this: If both parents are true, “Monkey” will have a 100% outcome to be true. If only one of its parents is true, the outcome is 50% to be true. Finally, “Monkey” will have a 0% outcome to be true if both parents are false together.

The probabilities of this situation can be calculated with the formula of a BBN. All three nodes, the variables have two possible values, T (for True) and F (for False). The joint probability function is:

$$P(M, Fl, C) = P(M|Fl, C)P(Fl)P(C)$$

where the names of the variables have been abbreviated to $M = \text{Monkey}$, $Fl = \text{fourLegged}$, and $C = \text{climbingTree}$.

The joint probability function can answer questions such as “What is the likelihood that “Monkey” is true, given that “fourLegged” is true and “climbingTree” is false?” by using the joint probability formula:

$$P(M = T | Fl = T, C = F) = \frac{P(M = T, Fl = T, C = F)}{\sum_{M \in \{T, F\}} P(M, Fl = T, C = F)} = \frac{0.5}{0.5 + 0.5} = 0.5.$$

From the result of the calculation, given the condition that “fourLegged” is true and “climbingTree” is false, it can be estimated that the child node, “Monkey” should have the 50% posterior probabilities to be true. Other pairs of parent and child nodes can be also processed with the joint probability formula to get the posterior probabilities.

3. CHARACTERISTIC VECTORS

In this section, we describe some previous approaches for software classification which inspired the idea of characteristic vectors and define characteristic vectors.

3.1 Software Classification

There are different types of software: coding language, application, target environment, and other classification factors. For operational environment adaptation or quality evaluation purposes, there should be referential rules for classifying each different type of software. There have been a few approaches regarding such classification. As one of

international standards, ISO/IEC TR 12182 [21] provides guidance for software categorization for proper interpretation and application of ISO (International Organization for Standardization) documents. The software categorization approach of ISO/IEC TR 12182 is primarily based on the multiple views of software classification.

Generally, quality evaluators or certification institutions can select essential views from the views of ISO/IEC TR 12182 to form their own quality certification model. For example, if the “User class” view is selected as the software domain of classification, the quality evaluators can have sub-domains of the “User class” view such as “General user” and “Expert user.” Finely classified sub-domains of the views can have the same effect as that of describing the characteristics of software. If there is a software product certified under the “Expert user” view, the warranty of the product is void when used by a “General user.” Although ISO/IEC TR 12182 can be used to set the effective range of quality evaluation or certification, there is no mechanism provided to extend the views.

Just like the views, there has also been another approach proposed such as the product-based software certification framework [5]. The approach is fundamentally based on the concept of “product-based certification” which is different from independent verification and validation (IV&V). In contrast to IV&V which occurs during each phase of the software development process, product-based certification occurs after the software product is built in order to determine whether the software product will satisfy its functional, performance, and quality expectations. The author [5] of the product-based software certification framework insists that product certification is better than other quality assessment methodologies, such as process maturity assessment and personnel accreditation. The methodologies of process maturity assessment and personnel accreditation lack measurable, objective data while the approach of product-based certification can produce repeatable and reproducible results which will produce the same result even if a different certifier or quality evaluator is engaged.

The product-based software certification framework consists of eight dimensions. The purpose is to create a set with one member from each dimension. The set represents a particular type of software. A certification methodology can be established for the particular type. The eight dimensions are:

1. Safety, Security, Reliability, Availability, Fault Tolerance, Performance;
2. Hard Real-time, Soft Real-time, Time Indifferent;
3. Embedded, Desktop, Mainframe, Internet;
4. “In a vacuum”, “With respect to the environment”;
5. Avionics, Medical, Finance, Telecom, Automotive, Nuclear, Maritime;
6. Executable Format, Source Code Format;
7. Custom Software, COTS;
8. Previous Version Certified, Never Certified.

For example, Security, Soft Real-time, Internet, “In a vacuum”, Telecom, Executable Format, Custom Software, and Previous Version Certified, can be one of several possible permutations. Each member of a set suggests what technology is needed to certify software defined by that set. This framework can be surely a kind of a set of good referential rules for the quality evaluators or certifiers although there is a lack of concern regarding extending each dimension of the framework.

3.2 Characteristic Vectors

In this paper, each unique COTS software product is represented with characteristic vectors which have components showing detailed properties of COTS software type. Unlike previous similar approaches, a characteristic vector can include characteristic sub-vectors as its components recursively. The value of a characteristic vector of COTS becomes 1, if the COTS has corresponding properties, otherwise it is 0. The formal definition of a characteristic vector is as follows.

CV: Characteristic Vector. $Val(i)$: value of characteristic vector i . (3)

$CV = (Val(CV_1), Val(CV_2), \dots, Val(CV_l))$, l is the dimension of CV .

$CV_i = (Val(CV_{i1}), Val(CV_{i2}), \dots, Val(CV_{im_i}))$, $1 \leq i \leq l$, m_i is the dimension of CV_i .

$CV_{ij} = (Val(CV_{ij1}), Val(CV_{ij2}), \dots, Val(CV_{ijn_{ij}}))$, $1 \leq j \leq m_i$, n_{ij} is the dimension of CV_{ij} .

$$Val(CV_i) = \begin{cases} 1, & \text{if } \sum_{j=1}^{m_i} Val(CV_{ij}) \geq 1 \\ 0, & \text{otherwise} \end{cases}$$

where if CV_{ij} is a leaf node, $Val(CV_{ij})$ is assigned 0 or 1 according to its corresponding property.

Characteristic vector CV may include the data of other different characteristic vectors, such as CV_i or CV_{ij} , as its components recursively. It is also possible to include more characteristic sub-vectors that are finely diverged under the level of CV_{ij} . The components of a vector must be specified as scalar values. The characteristic vector CV acquires a “1” if at least one of its components has a value of “1.” Otherwise, CV gets a value of “0.” For specifying CV , we will use 1 and True as well as 0 and False interchangeably from now on. If the CV , itself, is a leaf node which does not have any sub-vectors, the value of “0” or “1” is assigned directly for the CV , itself. Ultimately, the components of a characteristic vector attempt to represent the characteristics of specific software. Fig. 2 shows the relationships among the components of characteristics vectors. For comprehension, the numerical representation of Fig. 2 is as follows.

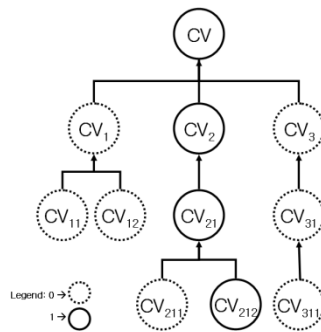


Fig. 2. Relationships among the components.

Table 1. Characteristic vector example.

Vector	Components
Application Software (CV_1)	Office Management (CV_{11})
	Financial (CV_{12})
	Manufacturing (CV_{13})
	Education (CV_{14})
	Credit Cards (CV_{15})
N/W & Communication (CV_2)	Internet (CV_{21})
	Resolution (CV_{22})
	WAP (CV_{23})
	Security (CV_{24})
	E-Mail (CV_{25})
H/W & System (CV_3)	O/S & System (CV_{31})
	Embedded (CV_{32})
	Monitoring (CV_{33})
	Load Management (CV_{34})
	Server (CV_{35})
Platform (CV_4)	UNIX (CV_{41})
	WINDOWS (CV_{42})

$$CV = (0, 1, 0)$$

$$CV_1 = (0, 0), CV_{11} = (0), CV_{12} = (0)$$

$$CV_2 = (1), CV_{21} = (0, 1), CV_{211} = (0), CV_{212} = (1)$$

$$CV_3 = (0), CV_{31} = (0), CV_{311} = (0)$$

For example, components of each characteristic vector are listed in Table 1. Table 1 is constructed based on TTAS.KO-11.0026 [22] which was referenced in setting up the software domains of the meta-model [10]. The formal representation of Table 1 is as follows.

$$CV_1 = (Val(CV_{11}), Val(CV_{12}), Val(CV_{13}), Val(CV_{14}), Val(CV_{15}))$$

$$CV_2 = (Val(CV_{21}), Val(CV_{22}), Val(CV_{23}), Val(CV_{24}), Val(CV_{25}))$$

$$CV_3 = (Val(CV_{31}), Val(CV_{32}), Val(CV_{33}), Val(CV_{34}), Val(CV_{35}))$$

$$CV_4 = (Val(CV_{41}), Val(CV_{42}))$$

In Table 1, every characteristic vector CV_i has CV_{ij} sub-vectors. For example, CV_{11} which is the component, "Office Management," belongs to CV_1 . Likewise, CV_{22} belongs to CV_2 as the component, "Resolution."

As mentioned earlier, if any components of CV_1 , such as CV_{11} , CV_{12} , CV_{13} , CV_{14} , and CV_{15} , correspond to any specific type of software, the component gets assigned the value, "1," which becomes the final value of CV_1 . Otherwise, the final value of CV_1 is "0." Depending on the state of the components, it can be said that CV_1 can be regarded as the fingerprint of software.

Let us consider another example. Assume that CV_2 is (0, 0, 0, 1, 1). This characteristic vector represents a specific type of COTS software products corresponding to a spe-

cific domain of software classification. According to the list in Table 1, the software characteristics corresponding to the components of CV_2 are “Security” and “E-Mail.”

Table 1 is an example of designating components for the characteristic vectors. The components can be set up with any criteria. Thus, by providing flexibility for certifiers of software quality, the characteristic vectors can be mapped into any prior format of software classification.

4. DERIVING AND USING DEPENDENCY RELATIONSHIPS

In this section, we describe a procedure of using a BBN for the relationships between the characteristic vectors and metrics.

4.1 Deriving Dependency Relationships

Each different COTS software product operates reliably in its own software domain. Thus, when there is a need to establish foundations of software quality evaluation, the certifier must make decisions on the specific metrics corresponding to the specific software domains.

To establish such domains, the approach of the grouping method in a top-down style was tried [10]. Each group of hierarchical software classification is assigned proper metrics for certification. Unlike the previous approach requiring additional works of metrics settlement, the main purpose of this paper is to describe automation of the allocation of metrics by integrating the structure of the BBN with the relationship network of characteristic vectors and metrics.

Based on the existing evaluation data of the Telecommunications Technology Association² (TTA) in Korea, we set up an example with six COTS software products. The initial data of the examples is organized and indexed in Table 2. To make it simple and clear, two types of COTS software are drawn from Table 1. That is, two characteristic vectors, CV_1 representing “Application Software” and CV_2 representing “N/W & Communication” become the basis of dividing the overall types of COTS software.

In Table 2, the external software quality metrics are specified in the parentheses following the symbol for the COTS software products. Those previous records of metrics application are very important and crucial for utilizing the existing data of past cases. Such fundamental data must be converted into normalized format first in order to be used. The normalized format used in this paper is the prior or conditional probability distribution of BBN.

In the BBN, the joint post probability must be calculated in order to forecast the posterior probability of an event occurrence regarding each node of the BBN. Many software tools for a BBN are normally used since manual calculation is very difficult. Fig. 3 shows the network of BBN nodes expressing the dependency relationships. The child nodes are the metrics having relationships which are shown as arcs in Fig. 3. The CV nodes are parent nodes representing the characteristic vectors. For example, the M_{10} node, which is the metric, “Interface standard compliance,” has two arcs coming from CV_{11} and CV_{24} because M_{10} was applied for the quality evaluation of a COTS software product corresponding to CV_{11} and CV_{24} . Likewise, other metrics nodes have arcs coming from

² <http://www.tta.or.kr/English>.

Table 2. An example of correspondence among prior data.

	CV_{12}	CV_{14}	CV_{11}	CV_{22}	CV_{24}
$P_1(M_1, M_3, M_8)$	•				
$P_2(M_{11}, M_{12})$		•			
$P_3(M_5, M_{11})$			•		
$P_4(M_2, M_6)$				•	
$P_5(M_4, M_7)$					•
$P_6(M_9, M_{10})$			•		•

* $P_1 \sim P_6$: COTS software products certified previously
 * $M_1 \sim M_{12}$: external software quality metrics used for prior certification
 M_1 : Functional adequacy
 M_2 : Functional implementation completeness
 M_3 : Functional implementation coverage
 M_4 : Functional specification stability
 M_5 : Computational accuracy
 M_6 : Data exchangeability
 M_7 : Access controllability
 M_8 : Access auditability
 M_9 : Functional compliance
 M_{10} : Interface standard compliance
 M_{11} : Failure resolution
 M_{12} : Incorrect operation avoidance

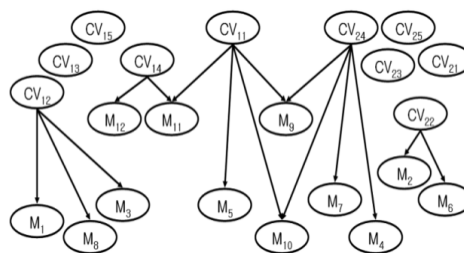


Fig. 3. A Bayesian belief network of characteristic vectors (CV) and metrics (M).

the parent CV nodes according to the past evaluation data. With this BBN, proper metrics can be estimated for a new COTS software product by adjusting the conditional probability distribution.

4.2 Constructing a Bayesian Belief Network

Besides the dependency relationships, a node of a BBN should have a conditional probability table that describes the possible status of parent nodes which are used to make a joint probability distribution table for actual inference. In this work, there are two possible statuses of parent nodes: “True” or “False.” If a child node has two parent nodes, there would be four possible statuses of the joint probability by the various combinations. Table 3 shows an example of a conditional probability table for the M_{10} node which is the metric, “Interface standard compliance,” as depicted in Fig. 3.

Table 3. The conditional probability distribution of M_{10} .

Parent Nodes (True = 1, False = 0)		Node: M_{10}
CV_{11}	CV_{24}	Probability of Being True
True	True	1.0
	False	0.5
False	True	0.5
	False	0.0

M_{10} is influenced by its two parents, CV_{11} and CV_{24} , which are the characteristic vector components, “Office Management” and “Security.” Each of the two parent nodes has two states, “True” or “False.” From this table, there are only four possible parent node states. Table 3 enumerates these possible four states as the prior probabilities of the BBN. If CV_{11} and CV_{24} are both true, M_{10} will have a probability of one of being true. If CV_{11} and CV_{24} are both false, then M_{10} has a probability of zero of being true. Otherwise, if only either one of the parent nodes becomes true, the probability is 0.5 for M_{10} .

After setting up the prior probability distribution, the BBN is ready to infer from various evidence of external events. Let us assume there is a new COTS software that needs to be evaluated. The software certifier wishes to know which metrics are suitable for the evaluation. The state of the characteristic vector of the new COTS product is as follows.

$$(CV_{11} = True, CV_{24} = True) \quad (4)$$

This vector means every node is false except CV_{11} and CV_{24} in the BBN. The values of this vector setting are applied as the evidence to evaluate the joint post probability distribution. From Fig. 3, the child nodes, M_4 , M_5 , M_7 , M_9 , M_{10} and M_{11} are influenced by the parent nodes, CV_{11} and CV_{24} . Considering the prior probabilities of the immediate parent nodes, the posterior probabilities of metric nodes can be expressed by:

$$\begin{aligned}
 P(M_1 = True \mid CV_{12} = True) &= 0.0 \\
 P(M_2 = True \mid CV_{22} = True) &= 0.0 \\
 P(M_3 = True \mid CV_{12} = True) &= 0.0 \\
 P(M_4 = True \mid CV_{24} = True) &= 1.0 \\
 P(M_5 = True \mid CV_{11} = True) &= 1.0 \\
 P(M_6 = True \mid CV_{22} = True) &= 0.0 \\
 P(M_7 = True \mid CV_{24} = True) &= 1.0 \\
 P(M_8 = True \mid CV_{12} = True) &= 0.0 \\
 P(M_9 = True \mid CV_{11} = True, CV_{24} = True) &= 1.0 \\
 P(M_{10} = True \mid CV_{11} = True, CV_{24} = True) &= 1.0 \\
 P(M_{11} = True \mid CV_{11} = True, CV_{14} = False) &= 0.5 \\
 P(M_{12} = True \mid CV_{14} = True) &= 0.0.
 \end{aligned} \quad (5)$$

M_{11} is noticeable since each parent node has a different evidence value. Furthermore, as half of the parent nodes are true, the posterior probability of M_{11} is set as 0.5. If one-

third of the parent nodes is true, the posterior probability would be 0.33. In this case, the metrics showing a higher posterior probability, such as M_4 , M_5 , M_7 , M_9 , and M_{10} , would be selected rather than other metrics for quality evaluation.

The crucial point of this simplified example is to describe the inference mechanism of our approach using a BBN in its key details. First of all, a BBN representing the relationships between the software characteristics and metrics must be constructed referring to the cases or reports of previous software quality evaluations. If there is a case showing that a software product having the “Security” characteristic was inspected by evaluation criteria of the “Access auditability” metric, the BBN node of “Access auditability” will be designated to have an arc coming from the node, “Security.” That means that “Access auditability” is under the influence of “Security.” Likewise, if there is another case showing that another software product of a “Monitoring” characteristic was inspected by the “Access auditability” metric, the BBN node, “Access auditability” will have the second arc coming from the node, “Monitoring.”

The BBN nodes connection is simple: a BBN node is connected by referring to the influencing factors. A node can be influenced by multiple parent nodes. In Fig. 3, we can see clearly that the connections between M_4 , M_5 , M_7 , M_9 , M_{10} , CV_{11} , and CV_{24} show that M_4 , M_5 , M_7 , M_9 , and M_{10} are under the influence of CV_{11} and CV_{24} . However, estimating the degree of influence is not that simple. Let us look at the nodes, M_9 and M_{10} . The nodes are both under the influence of characteristic vectors, “Office Management” and “Security,” simultaneously. If only one of the two characteristic vectors is emphasized for metrics selection, M_9 and M_{10} will have the outcome of 50% as the degree of importance. Likewise, if both characteristic vectors are designated together for metrics selection, M_9 and M_{10} will have a 100% degree of importance. Although this example is simple, handling and calculating the degree of importance is difficult for nodes of a BBN connected in a complicated manner. The automated BBN tools are necessary for actual operation.

Let us also take a look at the scenario of utilizing the inference result. The degree of importance is just the probability referred by the quality evaluators. Having the repository of a BBN representing the complex connection of nodes based on prior cases of quality evaluation, the quality evaluator adjusts the state of each of the characteristic vector nodes according to the evidence of external events. After such adjustments, the BBN shows the posterior probabilities, which are the importance degree of each metric, by updating the belief probability with the joint probability formula. From the example above, the quality evaluators must have test cases regarding the metrics, “Functional compliance” and “Interface standard compliance” if the outcome of inference shows that M_9 and M_{10} have a 100% degree of importance. Likewise, the quality evaluators need not have the test cases regarding the metric, “Functional compliance” if M_9 has only a 50% degree of importance. After utilizing the BBN, the operator of the BBN should pay attention to the results of the evaluation and use the new evaluation data to update the repository of the BBN.

5. DISCUSSION

In this section, we describe the reasons for using a Bayesian Belief Network and comparisons to other approaches.

5.1 The Reasons for using a Bayesian Belief Network

In this study, the BBN was selected because of the following reasons [23]: First, the lack of the past cases or training data does not pose a critical problem since the dependency relationship network can make inferences from other prior data. For example, an evaluator of software quality could miss setting up some of the nodes that are necessary to represent a specific type of COTS software product due to unclear and insufficient prior cases of evaluation. Despite the incomplete setting of nodes, BBN can infer the software type which is similar to the intended type if there are any dependency relationships between the node of the unclear software type and other nodes representing specific software types. Similarly, the problem, which is the unclear and inconsistent allocation of metrics existing in past cases, could be overcome by utilizing the learning and inferring capability of the BBN.

Second, the dependency relationship network conducts searches on a large area of software faults detection increasing the possibility of catching hard-to-find errors. Generally, quality evaluators apply metrics for the single domain of a specific application, ignoring other metrics belonging to other software domains. When inferring metrics with the BBN, quality evaluators can see the priority ranks of metrics in one big figure regardless of narrow domains. As a result, the chances of detecting the faults that are normally ignored are increased. For example, a BBN may designate and add the metric of data-overflow inspection for the COTS software which is assigned with specific metrics already whenever there is any change in the characteristic vectors showing a broader range of data processing.

Third, it is easy to integrate prior knowledge and newly learned experiences or data since the models of BBNs represent causality and probabilistic semantics. In the case of the approach utilizing past cases of software quality evaluation, setting the prior probability distribution of each node in the BBN might be regarded as the same as utilizing prior knowledge. Combining many instances of such prior knowledge, new data which is skipped or normally ignored, such as newly changed ranking of metrics priority in reverse order, can be obtained in the style of data-mining. Such newly obtained knowledge can be added to the existing BBN since operators of the BBN could alter or modify the structure of the nodes network easily.

In the prior approach [10], constructing a system of hierarchical software classification is required as the preceding work of metrics classification. Such a classification system is designed mainly by human evaluators, not based on formalized methodologies. Although the method proposed in this paper requires also the graph structure of a nodes network in a BBN as the preceding work, the nodes network is created naturally in a BBN, not by the intentions of human evaluators. Basically, the nodes of the BBN are connected to each other according to proper dependency relationships referring to past cases to bind related variables from the sample data. In this paper, the nodes of characteristic vectors, which are parent nodes representing the types of COTS software, influence the metrics nodes which are child nodes. Under such a structure of relationships, when the prior probability distribution of parent nodes changes according to the type of COTS software being evaluated, the joint post probability distribution of child nodes is calculated. Such a value of joint post probability can be regarded as representing the degree of importance for each child node to the metrics.

In software development processes and products, uncertainty, such as accuracy of requirements or completeness of testing, is inherent and inevitable [24]. Likewise, there is also the uncertainty regarding COTS software evaluation. To overcome the uncertainty, we have chosen the BBN because the existing past evaluation data is available and the dependency relationships can be extracted from the data. Moreover, the BBN incurs a lesser burden on software certifiers who are primarily interested in an actual field application rather than pure theory since the BBN only requires a small amount of knowledge of basic dependency relationships with respect to the specific software domains [25].

5.2 Comparing with Previous Studies

Although the theory of Bayesian probability, based on the BBN, has a long history. The BBN has been popular and recently has been adopted in many diverse fields as a result of improvements in algorithms and many software tools [26]. Many methodologies, such as genetic algorithms, artificial neural networks, or fuzzy algorithms, have some obstacles for handling complex models. For example, a model depicted in a BBN might not be expressed easily with the structure of neural networks since the basic architecture of neural networks should adhere to a canonical format. Thus, in a situation where specialized and non-canonical networking paths exist among the nodes, such as a case of characteristic vectors, must be kept, the BBN can be used to construct a network of nodes intuitively and visually.

As compared to previous studies [9, 13, 14], the clarified domains of quality metrics should be an advantage over these previous studies. In addition, with respect to development processes, the focus has been given to classifying metrics under the subjective view of certifiers without clearly specifying the types of COTS software. Therefore, when the approach, proposed in this paper, is applied to black-box testing, this approach takes the role of a decision making system.

Actually, whether a framework of software quality evaluation is performed for white-box testing or black-box testing, our approach can be applied and adapted for that framework to aid in deciding the coverable range of quality metrics. The main reason for assuming black-box testing is because COTS software products do not normally provide source code. Since our original research focused on COTS software products, we assumed basically that black-box testing would be the fundamental approach. The purpose of our approach is to provide guidelines for quality inspection based on a behavioral perspective of the software rather than to provide strictly enforced rules. Our approach is not concerned with the actual testing of COTS software products as it is just a referential guide. Quality certifiers can consult the inferred metrics of our approach to economically reduce the number of test cases. That is why our approach can be regarded as a kind of computer aided decision making.

Hence, a primary goal is to construct the network of characteristic vectors and metrics nodes corresponding to the specific domains of software classification regardless of having source code or development processes. The quality certifiers can also reflect their opinions and at the same time refer to the result of the joint post probability by easily adjusting the connection status of nodes and conditional probabilities. Such adjustment can make the result of inference react sensitively to the changes of sample data, which in turn enhances accuracy in decision-making.

Table 4. Comparisons among the approaches.

	Characteristic Vectors	[9]	[13]	[14]
Reflection of a quality evaluator's intention	Easy	Hard	Hard	Hard
Visual representation of nodes connection	Easy	Possible	Hard	Possible
Method of software quality evaluation	Black-box	White-box	White-box	White-box
Domain range of software classification	Narrow	Broad	Broad	Broad
Training of intelligent mechanism	Unnecessary	Needed	Necessary	Necessary
Support of domain experts	Unnecessary	Necessary	Necessary	Necessary

Table 4 shows a summarized comparison among other approaches, explored as related work and cited as reference, and our proposed approach shown as “Characteristic Vectors.” “Reflection of a quality evaluator’s intention” means the flexibility of changing the structure of intelligent mechanism such as the structure of network nodes connection. From Table 4, we see that it is hard to change the structure of the intelligent mechanisms except with our approach based on the BBN. “Visual representation of nodes connection” means easy manipulation of nodes consisting of the intelligent mechanism with automated software tools. Although there are many tools for the domain of genetic algorithms, neural networks, and fuzzy algorithms, the tools of the BBN provide the interfaces for handling each node that are easier than the tools for other approaches.

“Method of software quality evaluation” means whether the approach is based on black-box testing or white-box testing. Since our approach intends to be applied to COTS software products, black-box testing is appropriate. Other approaches are based on white-box testing because of the policy of inspecting source code. “Domain range of software classification” means whether the software classification policy is based on a narrow range or broad range. Since our approach requires detailed classification of software characteristics, the domain range must be narrow in contrast to the other approaches assuming a loose classification.

“Training of intelligent mechanism” means whether each approach requires a long time in training the intelligent mechanism or not. Unlike the other approaches, such as neural networks, requiring a separate and dedicated time for training, the structure of nodes connection in the BBN can be operated immediately after construction. Finally, “Support of domain experts” means whether each approach requires the support of experts or not in constructing the structure of the intelligent mechanism. In our approach, if there are enough prior cases of quality evaluations, we can construct the structure of the nodes connections referring to the previous cases of software characteristics and metrics. With other approaches, there should be an expert’s intervention to preserve the regular format of the intelligent mechanism.

On the other hand, there is a weakness of the BBN which should be considered regarding our approach. Each node of the BBN must have a conditional probability table according to the states of the parent nodes. The prior probabilities determining the possible outcomes of a node can be settled by the intention of the operator, not by strict rules. For example, an operator can settle the expected posterior probability of a node to be 50% when one of its two parents is set as “True” while another operator can set the ex-

Table 5. An extended example of characteristic vectors.

Vector	Components
CV_HW_and_System_Related (CV_1)	CVL_OS_and_System_Support (CV_{11})
	CVL_HW_or_System_Internal_Control (CV_{12})
	CVL_HW_or_System_External_Control (CV_{13})
	CVL_HW_and_System_Security_Support (CV_{14})
CV_NW_and_Communication (CV_2)	CVL_Internet_Utilization (CV_{21})
	CVL_ISDN_Utilization (CV_{22})
	CVL_NW_Resource_Management (CV_{23})
	CVL_NW_Security_Support (CV_{24})
CV_Application_SW (CV_3)	CVL_NW_Groupware (CV_{25})
	CVL_APP_Official_Business_Support (CV_{31})
	CVL_APP_Banking_Support (CV_{32})
	CVL_APP_Manufacturing_Support (CV_{33})
	CVL_APP_Education_Support (CV_{34})
CV_DB (CV_4)	CVL_APP_Security_Support (CV_{35})
	CVL_DB_Query_Processing (CV_{41})
CV_Development_Tool (CV_5)	CVL_DB_Data_Warehousing_Support (CV_{42})
	CVL_Computer_Program_Compile_and_or_Translation (CV_{51})
	CVL_Test_Support (CV_{52})
	CVL_CASE_Tool (CV_{53})
CV_etc (CV_6)	CVL_WEB_Development (CV_{54})
	Not Defined

characteristic vectors can be extended and mapped into any prior format for software classification freely and flexibly.

For the experiment, thirty records from past evaluations were used. Among the records, twenty evaluation records of COTS software were analyzed with a public software tool, called “GeNIe 2.0³,” to construct the BBN for the case study. The remaining evaluation records were used to validate the BBN of case study.

Table 6 shows the names of previously certified products and the software characteristics of each product in the form of characteristic vectors. For example, the product, “PK1_Ar” has the state of a characteristic vector: $CV_3 = (1, 0, 0, 0, 0)$. This state means the component of “CV_Application_SW,” “CVL_APP_Official_Business_Support” is set as true. Thus, it can be known that “PK1_Ar” is related to the field of business management. Meanwhile, in Table 6, any characteristic vector that does not always have specific components becomes true for the prior probability when applied to construct the BBN.

The overall architecture of the BBN can be described: The number of all of the nodes was about three hundred. The number of arcs connecting each node was about six hundred. The status of arcs connection must be preserved after analyzing the past cases when identifying the test metrics for the evaluations of new COTS software. The thickness of arc represented the strength of influences from the parent nodes to child nodes. The thicker arc meant a stronger influence. The nodes where the arrow head of arcs contacts are the child nodes.

³ <http://genie.sis.pitt.edu>.

Table 6. Product names of COTS software certified previously.

Product name	State of CV	Product name	State of CV
PK1_Ar	$CV_3 = (1, 0, 0, 0, 0)$.	PK16_Web	$CV_2 = (1, 0, 0, 0, 1)$. $CV_3 = (1, 0, 0, 0, 0)$.
PK2_Be_Project	$CV_1 = (1, 0, 0, 0)$. $CV_2 = (0, 0, 1, 0, 0)$.	PK17_RESORT	$CV_5 = (1, 1, 1, 0)$.
PK3_C	$CV_2 = (0, 0, 0, 1, 0)$.	PK18_Tower	$CV_4 = (0, 1)$. $CV_5 = (0, 0, 0, 1)$.
PK4_H	$CV_3 = (1, 0, 0, 0, 0)$. CV_2 .	PK19_One	$CV_3 = (0, 0, 0, 1, 0)$.
PK5_N	$CV_3 = (1, 0, 0, 0, 0)$. CV_2 .	PK20_MANTIS	$CV_1 = (0, 0, 1, 0)$. $CV_3 = (0, 0, 1, 0, 0)$. $CV_4 = (0, 1)$.
PK6_S	$CV_2 = (0, 0, 1, 1, 0)$. $CV_3 = (0, 0, 0, 0, 1)$.	PK21_SAFE	$CV_3 = (0, 0, 0, 0, 1)$.
PK7_AL	$CV_3 = (1, 0, 1, 0, 0)$.	PK22_Gallery	$CV_5 = (0, 0, 0, 1)$.
PK8_CL	$CV_3 = (0, 1, 0, 0, 0)$. $CV_2 = (1, 1, 0, 0, 0)$. CV_4 .	PK23_Card	$CV_3 = (1, 0, 0, 0, 0)$. $CV_4 = (0, 1)$.
PK9_NA	$CV_3 = (0, 0, 0, 1, 0)$.	PK24_Calc	$CV_3 = (1, 0, 0, 0, 0)$.
PK10_Si	$CV_1 = (0, 1, 0, 0)$. $CV_2 = (1, 0, 0, 0, 0)$.	PK25_SuS	$CV_1 = (1, 0, 0, 0)$.
PK11_B	$CV_3 = (0, 0, 0, 1, 0)$. $CV_2 = (1, 0, 0, 0, 0)$.	PK26_Professional	$CV_1 = (1, 0, 0, 0)$.
PK12_Na	$CV_2 = (1, 0, 0, 0, 0)$. $CV_3 = (0, 0, 0, 1, 0)$.	PK27_Stream	$CV_2 = (1, 1, 0, 0, 0)$.
PK13_Class	$CV_3 = (0, 0, 0, 1, 0)$.	PK28_Pa	$CV_1 = (1, 0, 0, 0)$.
PK14_TOS	$CV_2 = (0, 0, 0, 1, 1)$.	PK29_Blue	$CV_5 = (1, 1, 1, 0)$.
PK15_Nomix	$CV_3 = (1, 0, 0, 0, 0)$. $CV_2 = (1, 0, 0, 0, 0)$.	PK30_Flow	$CV_2 = (0, 0, 0, 0, 1)$. $CV_3 = (1, 0, 0, 0, 0)$.

In the BBN of case study, each node belonged to one of the three node groups. In Fig. 4, the nodes located in the upper position belonged to the group of characteristic vectors. The nodes located under the characteristic vectors represented each product of the COTS software evaluated previously. The nodes located at the bottom belonged to the group of metrics.

The nodes network of characteristic vectors was designed to represent the hierarchy of characteristic vectors under the architecture of parent and child relationship. The components consisting of each sub-vector influenced the higher vectors as parent nodes. For example, in Fig. 4, we can see that the node, “CV-HW_and_System_Related,” under the influence of four parent nodes: “CVL-OS_and_System_Support,” “CVL-HW_or_System_Internal_Control,” “CVL-HW_or_System_External_Control,” and “CVL-HW_and_System_Security_Support.” The posterior probability of the higher vector, “CV-HW_and_System_Related,” shall be set to be “True” if the evidence value of any parent node gets to be “True,” according to the conditional probability table of the BBN repre-

sending the architecture of characteristic vectors.

From the node of characteristic vector located at the top, such as “CV-HW_and_System_Related,” we could trace the influencing flow coming down from the node. Let us take another look at the other characteristic vector, “CV-NW_and_Communication,” in Fig. 4. From the node of the characteristic vector, two directed arcs coming down to make contacts with the nodes of previously certified products of COTS software, “PK4-H” and “PK5-N.” That kind of directed connection the COTS software products had the characteristics of network and communication. Because there also other parent nodes having different values of external evidence, the nodes, “PK4-H” and “PK5-N” 50% posterior probability to be true.

From the two nodes, “PK4-H” and “PK5-N,” there also directed arcs coming down to make contacts with the nodes of metrics such as “M-Functional_Implementation_Coverage,” “M-Computational_accuracy,” and “M-Data_Exchangeability_Data_Format_Based.” This that those three metrics were applied to the product, “PK4-H” and “PK5-N.” The current status of BBN, shown in Fig. 4, that a new product of COTS software having the characteristics of network and communication is about to be tested seeking adequate metrics for the criteria for a quality evaluation.

Having the evidence showing the characteristics of new software fed into the BBN, the node, “M-Data_Exchangeability_Data_Format_Based,” the highest posterior probability, 52% to be true, among the three nodes of metrics after calculating the joint probability function of many other different parent nodes. Highest posterior probability the metric of the node was frequently used for the certification of products corresponding to the parent nodes. However, the final decision of metric selection up to the quality certifier.

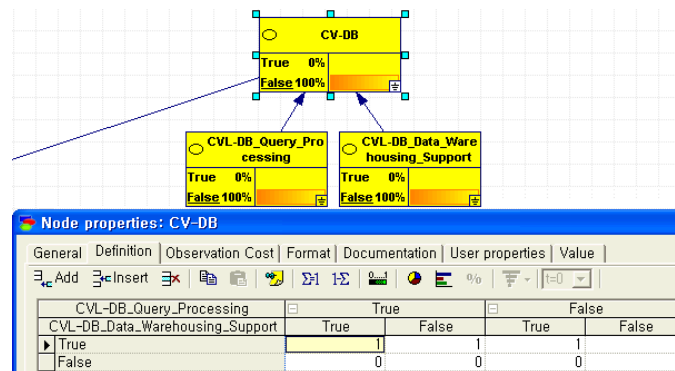


Fig. 5. Setting the conditional probability distribution using the tool.

Concerning the influence flow described above, setting up the conditional probability distribution of each node the most crucial point of constructing a BBN. Fig. 5 shows an example of conditional probability setting using the BBN tool, “GeNIe 2.0.” The node of the characteristic vector, “CV_DB,” under the influence of two parent nodes, which actually the components of characteristic vectors: “CVL_DB_Query_Processing” and “CVL_DB_Data_Warehousing_Support.” According to the definition of a characteristic vector, if at least one of the parent nodes becomes true, the posterior probability of “CV_

DB” should be 100% to be true. Otherwise, if the parent nodes both become false simultaneously, then “CV_DB” should have a 0% posterior chance to be true. The connection architecture of the other nodes, such as the nodes of the COTS software and metrics, is under the mutual influence of the conditional probability distribution.

In the case of nodes of the COTS software which connected as a child with the nodes of the characteristic vectors according to the prior cases of certification, the node connections should be regarded as the representations of software types. For example, in Fig. 4, the node, “PK2-Be_Project,” connected with the node, “CVL-OS_and_System_Support.” This that the COTS software product, “PK2-Be_Project,” has characteristics of supporting operating systems and general systems. In Fig. 4, the “CVL-OS_and_System_Support” node set as “True,” so the posterior probability of “PK2-Be_Project” calculated as 100%, which a strong influence on specific software characteristics.

The nodes located under the nodes of COTS software the metrics nodes. As one of the metric nodes which under the influence of COTS software nodes, “M-Functional_Adequacy” connected as a child with “PK2-Be_Project” by referring to the past cases of certification. After setting up the prior conditional probabilities and evidence values corresponding to a new COTS software product which required selection of test metrics, we found that the appropriateness of the metric node, “M-Functional_Adequacy” calculated as 60% as shown in Fig. 4.

Up to this point, the simplest description of the connection flow can be shown for the BBN of this case study: characteristic vectors \rightarrow COTS software \rightarrow metrics. We assert that the metrics nodes located at the bottom reflect the result of complex processing for the evidence of external events which were fed into the network representing the dependency relationships which were realized by the conditional probability distribution.

Although there could be many parent nodes per each node, setting up prior conditional probabilities using a BBN tool is relatively simple. For example, twelve parent nodes representing the previously certified products of COTS software are applied to determine the prior conditional probabilities of the metric node, “M-Computational_accuracy.” A conditional probability value, such as 0.882353, represents a specific condition having “True” parent nodes of more than ten nodes which are the COTS software nodes. The key point is the ratio of parent nodes having the prior probability of being “True.” In this case, 10/12 nodes are set as true.

The public tool of BBN, GeNIe 2.0, has a function called “diagnosis.” The diagnosis function shows the ranks of nodes, which are designated as “target,” according to the size of the joint post probability. In this paper, such ranks can be regarded as the priority levels of metrics which shall be applied first for the evaluations of new COTS software products. Fig. 6 shows the sorted ranking of inferred metrics for a new product of COTS software corresponding to the true nodes of characteristic vectors as follows: $CV_2 = (0, 0, 1, 0, 0)$ and $CV_1 = (1, 0, 0, 0)$, meaning the product is related to the characteristics of network management and operating systems. The inferred metric, such as “M-Mean_Time_Between_failures_MTBF,” listed at a higher rank with a probability of 0.857 (85.7%). Such metrics having high posterior probabilities shall be applied as first priority for tasks of quality evaluations.

In contrast, “M-Audit_Trail_Capability” shall be applied selectively upon the decisions of quality evaluators since the metric node has low probability, 0.286 (28.6%). In brief, the probability ranking list of the BBN tool can be served as a guideline for quality

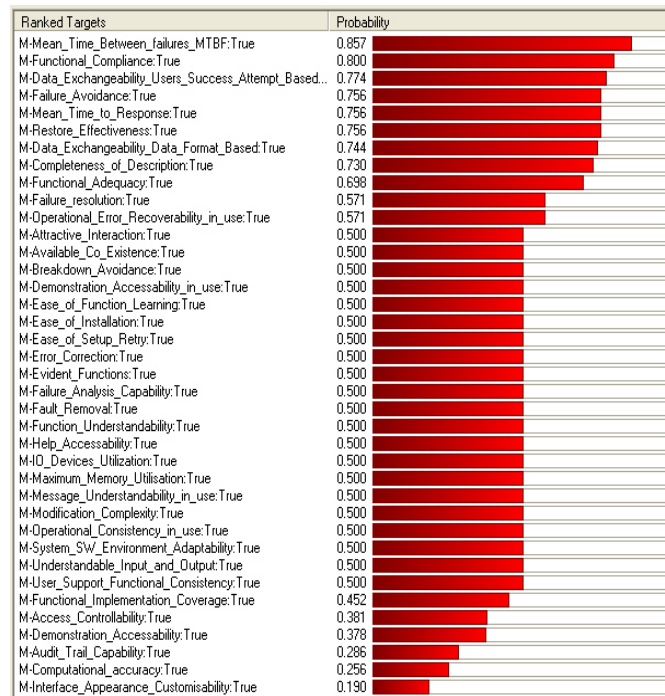


Fig. 6. Joint post probability ranking of the metrics nodes.

evaluation. This could be accomplished under the following principle: the higher posterior probability, the first priority metric.

After constructing this BBN for the case study, a validation was performed with nine prior cases of COTS software evaluation which were not used for the construction of the initial BBN. In detail, the validation was accomplished with the two different values of the posterior probability threshold for the metrics selected by the BBN. At the 50% threshold of posterior probability, the metrics inferred with the BBN were selected according the sorted order of posterior probability ranging from the top 100% to a 50% marginal threshold. Under the 50% marginal threshold, the metrics were considered irrelevant to the specific characteristics of the COTS software. Likewise, at the 30% threshold, the metrics inferred with the BBN were selected according the posterior probability ranging from the top 100% to a 30% marginal threshold.

After inferring the metrics with the BBN, the result was compared with the evaluation records of corresponding COTS software, which were prepared by human experts, to measure the inferential capability of the BBN. In this validation, the 50% marginal threshold was meant to test the BBN in a pessimistic situation while the 30% marginal threshold represented an optimistic situation. Table 7 shows the success ratios of estimating the appropriateness of metrics with the BBN. In Table 7, the success ratio is defined as follows.

$$\text{success ratio} = \frac{n(B \cap T)}{n(B \cup T)}$$

where

$n(S)$ is the number of elements in set S ,

B represents the set of inferred metrics with 30% or 50% threshold in BBN ,

T represents the set of metrics selected by human experts in TTA .

Due to the slight differences of preferences settled by the certifiers of TTA selecting the metrics, and the setting of conditional probability tables in BBN which was done by our own judgment, the result of Table 7 could fluctuate for different situations. However, we were certain that the BBN can simulate the mechanism of human intelligence in a simple way.

Table 7. Success ratio of estimating the appropriateness of metrics.

Product name	PK21	PK22	PK23	PK24	PK25	PK26	PK27	PK28	PK29
Success ratio at 30% threshold	83%	81%	72%	79%	82%	70%	64%	86%	90%
Success ratio at 50% threshold	79%	78%	69%	79%	82%	73%	64%	83%	87%

7. CONCLUSION

This paper proposed the use of a BBN to analyze the existing software quality evaluation data, based on the idea of inductive inference. The main objective has been to derive dependency relationships between the software characteristics and metrics in the BBN. The merit of this method is the probabilistic inference which is suitable for handling vague relationships of sample data. Another advantage is that it is easy to maintain as a BBN since there are many tools for BBNs available.

It is not easy to say that our approach gives a better, more accurate estimation than other approaches. If there are plenty of prior cases of quality evaluations, our approach might produce useful results of metrics estimation since a BBN is constructed with reference to the data of prior cases. Likewise, other approaches based on the mechanism of inductive inference could also produce useful results for estimation when their repository is full of prior cases data. Thus, it is not easy to determine the accuracy of estimation comparing to others when the exactly same environment is not available for each approach. If the advantages of our approach are emphasized, "hierarchy" and "flexibility" should be candidates for prime emphasis. The hierarchy structure forming among the nodes of characteristic vectors and metrics provides the foundation of an easy extension for the future data of quality evaluations in a systematic way. Moreover, the simple mechanism of a nodes network extension can be operated visually by users assuring the fast growing of BBN and data repository, which means the accuracy of estimation can get better in the future.

If software evaluation institutions adopt the method proposed in this paper, we can expect some benefits. A BBN can express the appropriateness of specific metrics in probabilistic detail for specific software and not just in a form of Yes or No. This would increase the potential of finding trivial faults since any metrics with probability greater

than zero would be inspected. The metrics priorities based on joint post probability distribution could also facilitate establishment of a roadmap for software evaluation or evaluation plans.

In this paper, the prerequisite of constructing the initial BBN is setting up the prior conditional probabilities for each node. While there are no clear criteria to settle on such prior probabilities, there could be vague selections of probabilities by human evaluators. Such vagueness could be resolved having a consistent method of probabilities settlement by an accumulating extensive amount of prior evaluation cases gradually. Another alternative resolution is publishing a guideline of setting probabilities under cooperation of domain experts.

In addition, we can consider the possible fluctuation of inferential outcomes produced by the BBN. A BBN requires a substantial amount of attention for maintenance of diverse operators for versatile applications. While adding nodes, altering the network, or modifying the prior conditional probabilities, the outcome of inference can be totally different from that of a previous application when inferring for the metrics of new COTS software which is very similar to that of previous software. Despite these possibilities of fluctuation of inference that can cause confusion, the BBN has the capability of coping with noisy data. If the operators let BBN evolve itself to handle such difficult situations, the impacts of problems could be minimally reduced.

In the future, we plan to define varied rules for setting up the prior probabilities. At this time, the prior probabilities of each node are set up by merely referring to the average number of its parent nodes. An experiment with large scale field data could be very fruitful. The public data of software reviews which are available on the Internet can be used to set up the BBN since the software evaluation institutions do not tend to disclose their software evaluation records to the general public. The outcome of Bayesian inference will be compared to that of human software evaluation experts to examine the various possible capabilities of the BBN.

REFERENCES

1. IDC, *Worldwide IT Spending 2006-2010 Forecast*, The Worldwide Black Book, Vol. 2, 2006.
2. I. Sommerville, *Software Engineering*, 7th ed., Addison Wesley, New York, 2004.
3. M. A. Friedman and J. M. Voas, *Software Assessment: Reliability, Safety, Testability*, John Wiley & Sons, Inc., New York, 1995.
4. C. Jones, *Software Assessments, Benchmarks, and Best Practices*, Addison-Wesley, New York, 2000.
5. J. Voas, "Limited software warranties," in *Proceedings of the 7th IEEE International Conference and Workshop on the Engineering of Computer Based Systems*, 2000, pp. 56-61.
6. J. Voas and K. Miller, "Software certification services: encouraging trust and reasonable expectations," *IT Professional*, Vol. 8, 2006, pp. 39-44.
7. ISO/IEC: DTR 9126-2, *Software Engineering – Product quality Part 2 – External metrics*, ISO/IEC JTC1/SC7 N2419, 2001.
8. V. R. Basili and B. Boehm, "COTS-based systems top 10 list," *IEEE Computer*, Vol.

- 34, 2001, pp. 91-95.
9. S. Dick and A. Kandel, "Fuzzy clustering of software metrics," in *Proceedings of the 12th IEEE International Conference on Fuzzy Systems*, Vol. 1, 2003, pp. 642-647.
 10. J. Oh, B. Lee, D. Park, J. Lee, E. Hong, and C. Wu, "Using hierarchical classification to certify software packages," in *Proceedings of the 1st ACIS International Conference on Software Engineering Research and Applications*, 2003, pp. 270-275.
 11. C. Lee, J. Oh, B. Lee, and C. Wu, "Selecting metrics for package software certification using formal concept analysis," in *Proceedings of the 2nd International Conference on Software Engineering Research and Applications*, 2004, pp. 291-297.
 12. F. V. Jensen, *An Introduction to Bayesian Networks*, Springer Verlag, New York, 1996.
 13. R. A. Vivanco and N. J. Pizzi, "Identifying effective software metrics using genetic algorithms," in *Proceedings of Canadian Conference on Electrical and Computer Engineering*, Vol. 2, 2003, pp. 1305-1308.
 14. M. D. Alexiuk and N. J. Pizzi, "Discriminatory software metric selection via a grid of interconnected multilayer perceptrons," in *Proceedings of IEEE Canadian Conference on Electrical and Computer Engineering*, Vol. 2, 2003, pp. 1131-1134.
 15. N. E. Fenton and M. Neil, "A critique of software defect prediction models," *IEEE Transactions on Software Engineering*, Vol. 25, 1999, pp. 675-689.
 16. M. Neil and N. E. Fenton, "Predicting software quality using Bayesian belief networks," in *Proceedings of the 21st Anniversary Software Engineering Workshop*, 1996, pp. 217-230.
 17. N. Fenton, P. Krause, and M. Neil, "Software measurement: uncertainty and causal modeling," *IEEE Software*, Vol. 19, 2002, pp. 116-122.
 18. A. Rajan, "Coverage metrics to measure adequacy of black-box test suites," in *Proceedings of the 21st IEEE/ACM International Conference on Automated Software Engineering*, 2006, pp. 335-338.
 19. L. H. Tahat, B. Vaysburg, B. Korel, and A. J. Bader, "Requirement-based automated black-box test generation," in *Proceedings of the 25th Annual International Computer Software and Applications Conference*, 2001, pp. 489-495.
 20. M. T. Mitchell, *Machine Learning*, McGraw-Hill, 1997.
 21. ISO/IEC, "FCD 12182 information technology – Categorization of software," ISO/IEC JTC1/SC7 N1949, 1998.
 22. Telecommunications Technology Association (TTA), Standard for a classification scheme of software reuse, TTAS.KO-11.0026, 2000.
 23. D. Heckerman, "A tutorial on learning with Bayesian networks," Technical Report MSR-TR-95-06, Microsoft Research, Redmond, WA, 1995.
 24. H. Ziv and D. Richardson, "Constructing Bayesian-network models of software testing and maintenance uncertainties," in *Proceedings of International Conference on Software Maintenance*, 1997, pp. 100-109.
 25. N. E. Fenton, M. Neil, and A. Finkelstein, ed., *Software Metrics: Roadmap, The Future of Software Engineering*, ACM Press, New York, 2000.
 26. I. Ben-Gal, *Bayesian Networks, Encyclopedia of Statistics in Quality and Reliability*, John Wiley & Sons, New York, 2007.



Chongwon Lee (李鍾源) received the B.S. degree in Computer Science and Information Engineering from Yonsei University, Seoul, Korea in 2001, and the M.S. and Ph.D. degrees in Computer Science from Seoul National University, Seoul, Korea in 2003 and 2008, respectively. Currently, he is a research professor of the School of Computer Science at the University of Seoul, Korea. His research interests include testing, evaluation and certification of computer software products.



Byungjeong Lee (李秉政) received the B.S., M.S., and Ph.D. degrees in Computer Science from Seoul National University in 1990, 1998, and 2002, respectively. He was a researcher of Hyundai Electronics, Co. from 1990 to 1998. Currently, he is an associate professor of the School of Computer Science at the University of Seoul, Korea. His research areas include software reengineering, web engineering and testing, evaluation and certification of computer software products.



Jaewon Oh (吳在原) received the B.S., M.S., and Ph.D. degrees in Computer Science from Seoul National University in 1997, 1999, and 2004, respectively. He was a senior researcher of the Mobile Software Platform Team in Samsung Electronics from 2004 to 2007. Currently, he is a full-time instructor of the School of Computer Science and Information Engineering at the Catholic University of Korea. His current research interests include mobile platforms, the objects technology, and software certification.



Chisu Wu (禹治水) received the B.S. degree in Applied Mathematics from Seoul National University in 1972 and the M.S. and Ph.D. degrees in Computer Science from Seoul National University in 1977 and 1982, respectively. He was a visiting researcher at the Loughborough University in 1978. From 1975 to 1982, he was an associate professor in the Department of Computer Science at Ulsan University, Korea. Currently, he is a professor in the School of Computer Science and Engineering at Seoul National University, where he has been since 1982. His research interests include software engineering and programming language. He is a member of the ACM.