Quality Prediction and Assessment for Product Lines

Hongyu Zhang, Stan Jarzabek, and Bo Yang

Department of Computer Science
School of Computing
National University of Singapore
3 Science Drive 2, Singapore 117543
{zhanghy, stan, yangb}@comp.nus.edu.sg

Abstract. In recent years, software product lines have emerged as a promising approach to improve software development productivity in IT industry. In the product line approach, we identify both commonalities and variabilities in a domain, and build generic assets for an organization. Feature diagrams are often used to model common and variant product line requirements and can be considered part of the organizational assets. Despite their importance, quality attributes (or non-functional requirements, NFRs) such as performance and security have not been sufficiently addressed in product line development. A feature diagram alone does not tell us how to select a configuration of variants to achieve desired quality attributes of a product line member. There is a lack of an explicit model that can represent the impact of variants on quality attributes. In this paper, we propose a Bayesian Belief Network (BBN) based approach to quality prediction and assessment for a software product line. A BBN represents domain experts' knowledge and experiences accumulated from the development of similar projects. It helps us capture the impact of variants on quality attributes, and helps us predict and assess the quality of a product line member by performing quantitative analysis over it. For developing specific systems, members of a product line, we reuse the expertise captured by a BBN instead of working from scratch. We use examples from the Computer Aided Dispatch (CAD) product line project to illustrate our approach.

1 Introduction

The challenge in today's software engineering is to deliver high-quality software on time to customers [33]. Successful companies must have a focus on customer satisfaction and software quality to ensure that the desired quality is built into the software product and that customers remain loyal to the company. This is especially true for the IT industry where customers have ever-increasing expectations of software quality. Software quality has become a major concern of software organizations [43]. However, it is well known that quality is "hard to define, impossible to measure, easy to recognize [26]". A lot of research thus has been done to refine the concept of quality into a number of *quality attributes* (also known as quality characteristics, quality factors or quality criteria (see e.g., [1, 7, 15, 36]). The ISO 9126 standard for information technology [19] provides a framework for the evaluation of software quality, which defines six product quality characteristics, i.e., functionality, reliability, usability, efficiency, maintainability, and portability. Nevertheless, it is a general consensus that different quality attributes can be adopted based on different users and their different intentions of the use of the software. In the software engineering context, quality attributes are also called non-functional requirements (NFRs) [9, 28, 32, 39].

Software product line (also called product family or system family) approach, initiated by Parnas [34], has attracted a lot of attention from both researchers and practitioners [11, 42]. It proves to be an effective approach to improve software development productivity [2, 29] and it has been successfully applied to the development of IS software. A product line arises from situations when we need develop multiple similar products for different clients, or from a single system over years of evolution. Members of a product line share many common requirements and characteristics. They may perform similar tasks, exhibit similar behavior, or use similar technologies. While having much in common, they still differ in certain requirements, design decisions, and implementation details. The variability stems from many sources such as customer's specific needs, mutability of the environment, system maintenance and evolution, and so on.

In the product line approach, we identify both commonalities and variabilities in a domain, and build generic assets such as domain model, product line architecture, generic components, etc. During reuse-based application engineering (the process of producing specific product line members), we reuse the product line assets instead of working from scratch. Generally speaking, there could be a large number of variants in a product line and not all configurations of the variants are valid or "good" in terms of the quality attributes of the target system. Some variants, especially design decisions, have considerable impact on system quality. Very often, we need to make tradeoffs among many "competing" decisions based on the achievable system quality and thus a systematic way of decision-making among different configurations is needed.

Feature diagrams [24] are commonly used in modeling common and variant product line requirements. Feature models appeal to many product line developers because customers and engineers usually speak of product characteristics in terms of the features the product has or delivers, so it is natural and intuitive to express any commonality or variability in terms of features [25]. A feature diagram provides a graphical tree-like notation that shows the hierarchical organization of the features. By traversing feature trees, we can find out which variants have been anticipated and accommodated in a product line. However, a feature diagram alone does not show how different configurations affect system quality attributes.

In this paper, we propose a Bayesian Belief Network (BBN) based approach to explicitly modeling the impact of variants (especially design decisions) on system quality attributes. A BBN represents domain experts' knowledge and experiences accumulated from the development of similar projects. It helps us capture the impact of variants on quality attributes, and helps us predict and assess the quality of a system by performing quantitative analysis over it. The constructed BBN is an important reusable asset in an organization. With the help of the BBN model, during reuse-based application engineering we can select a configuration of variants in a more informed and rational way.

As a representation of design knowledge, the constructed BBN could be stored and managed in an experience factory [4], which facilitates accumulation of project experiences in order to continuously improve software development practices. In this way, our approach helps companies to progress from ad hoc development towards the "rational design process".

In the remaining part of the paper, we describe our approach and illustrate it with examples from our domain-engineering project on the Computer Aided Dispatch (CAD) domain.

2 CAD Domain Overview

In this section, we give a brief overview of CAD domain. Computer Aided Dispatch (CAD) systems are mission-critical systems that are used by police, fire & rescue, health service, port operation company and many others. Figure 1 depicts a basic operational scenario and roles of a CAD system for Police.

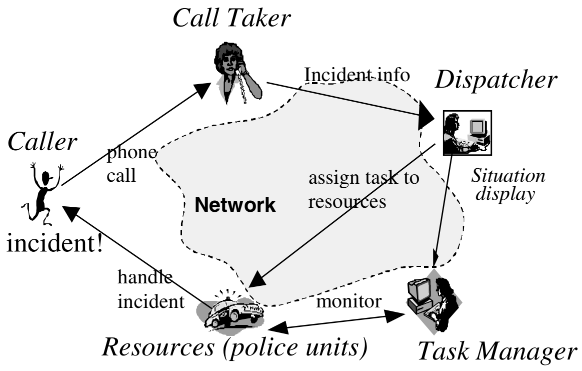


Fig. 1. The basic operational scenario in a CAD system for Police

Once a Caller reports an incident, a Call Taker in command and control center captures the details about the incident and the Caller, and creates a task for the incident. The system shows the Dispatcher a list of un-dispatched tasks. The Dispatcher examines the situation, selects suitable Resources (police units) and dispatches them to execute the task. The Resources carry out the task instructions and report to the Task Manager. The Task Manager actively monitors the situation and at the end, closes the task.

CAD systems in general shall exhibit the following quality attributes: high performance (in response time), high security, high scalability and low cost.

At the basic operational level, all CAD systems are similar - they support the dispatch of units to handle incidents. However, there are also differences across CAD systems. All these differences can be treated as variant requirements for developing a CAD product line. After domain analysis, we have identified the following variants in a CAD product line:

1. Architectural style: choosing between a three-tier one and a distributed one. In a three-tiered (client-business logic-database) system, business logic and database transactions are done in servers at control center. In a fully distributed system, the

business logic is distributed among different nodes (such as Call Taker, Dispatcher, Task Manager); each node is an active object that communicates with other nodes. A distributed system is expected to improve system performance and scalability, but may incur higher cost.

2. Database: The database can be a centralized database that is stored in a central place. Alternatively, it can also be a distributed database, which allows data to be distributed according to the organization's structure. For example, a distributed database allows the Caller Information table to be stored closer to the Call Taker site. As the distributed database allows the access of the data closer to the data usage, the communication traffic can be reduced and the performance can be improved. However, the distributed database may require more design and development effort, thus it may increase system cost. Furthermore, the selection of distributed database is dependent on the selection of architectural style – only when the distributed architectural style is selected can the distributed database be selected.
3. Deciding whether the system is to be Web-based or not. In a Web-based system, clients are web clients (based on Web browsers), communicating with server via the Internet/Intranet. A Web-based system is expected to be more scalable, however, it could involve more security concerns.
4. Deciding whether or not to encrypt the data that is transmitted between the clients and the server. Encrypting the data has negative impact on the system performance, but it improves security.
5. Selecting wireless modem (9600 bps or 19200 bps) for the communications between the control center and the mobile Resources. Wireless modems are connected to mobile Resources' MDT (Mobile Data Terminal), some wireless networks can only support data rates of up to 9600 bps. Selecting 9600 bps modem may cause lower performance and lower cost. Selecting 19200 bps modem may cause higher performance but also higher cost.
6. Validation of caller and task information. In some CAD systems, a basic validation check (i.e., checking the completeness of the Caller and Task info) is sufficient; in other CAD systems, validation includes duplicate task checking, VIP place checking, etc.; yet in other CAD systems, no validation is required at all.
7. Call Taker and Dispatcher roles (referred to as CT-DISP variant). In some CAD systems, Call Taker and Dispatcher roles are separated (played by two different people), while in other CAD systems the Call Taker and Dispatcher roles are played by the same person.
8. Dispatch Algorithm. There are different ways to dispatch Resources, using shortest distance search algorithm or location code search algorithm.

3 Modeling Variants in CAD Product Line by a Feature Diagram

Feature diagrams [24, 25] are often used to model common and variant product line requirements. Figure 2 shows a fragment of a feature diagram for CAD product line. Tools such as AmiEddi (available at www.generative-programming.org) can be used to construct feature diagrams.

Feature diagram provides a graphical tree-like notation that shows the hierarchical organization of the features. By traversing the feature trees, we can find out which variants have been anticipated during domain analysis. Features are classified as mandatory, optional and alternative (Czarnecki and Eisenecker also proposed the or-features [12]). Common requirements can be modeled as mandatory features whose ancestors are also mandatory. Variant requirements can be modeled as optional, alternative, or or-features. For example, the "Call Taker and Dispatcher roles" requirement described above has two alternative variants: "Separated" and "Merged". The optional "Validation" requirement has two or-variants: "Basic Validation" and "Advanced Validation", which means that the "Validation" requirement can be "Basic Validation", "Advanced Validation", or both or neither of them.

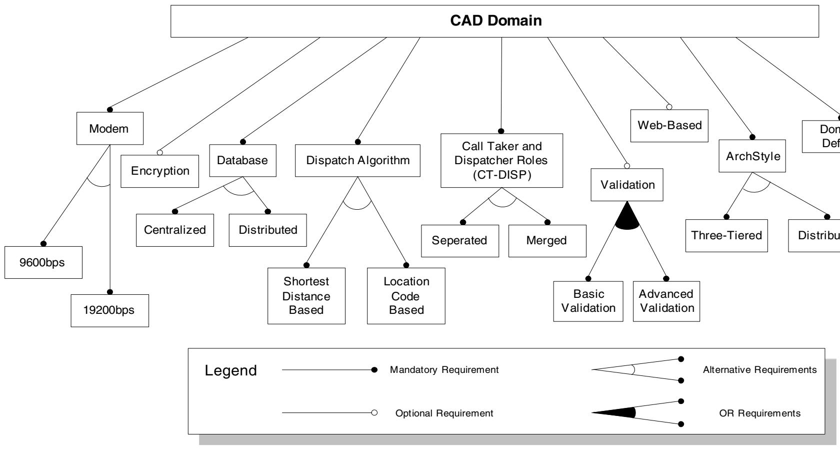


Fig. 2. CAD feature diagram (partial)

From a feature diagram, we can select a configuration of variants to construct a specific system (a product line member). For example, we can choose the following configuration of variants:

19200 bps modem, no encryption, centralized database, location code based dispatch algorithm, merged Call Taker and Dispatcher roles, basic validation, non-web based system, three-tired architecture

Obviously there exist many other configurations. Different configurations may lead to different systems that satisfy the same set of functional requirements. A natural question arises, i.e., how to choose among all possible configurations? Is there a systematical way by which we can select a "good" configuration under some criteria to construct the target system? Feature diagram alone does not give the answer or any guidelines to this question.

4 Modeling the Impact of Variants on System Quality

Generally speaking, besides the functional requirements, the customer is also very much concerned about the quality of the target system, such as performance, reliability, scalability, etc. Therefore, it is essential to assess the impact of different configurations on the system quality. If a serious negative impact of one configuration on some of the system quality attributes has been detected, we shall consider not to choose this configuration. In fact, we should aim at a target system that exhibits most desired quality attributes by choosing a "good" configuration.

Unfortunately, it is not an easy job to assess (either qualitatively or quantitatively) the impact of different configurations on the quality attributes of the target system. Basically, the difficulty results from the illusive nature of the quality and is also due to the complex interrelationships among variants (especially design decisions) and quality attributes. Taking the CAD product line as an example, we can see that:

- One variant may influence many quality attributes. For example, the choice of using data encryption has influence on both performance and security.
- One quality attribute may be influenced by many variants. For example, the performance quality attribute is influenced by the choice of database, architecture style, modem, encryption, and so on.
- Different variants may be competing or synergic. One quality attribute may be affected by many variants, with some contributing positively and others contributing negatively.

Not only the relationship among variants and the quality attributes of the target system is complex, but there is also some degree of uncertainty involved [13, 32]. In fact, there are so many factors that have impact on the quality attributes, one cannot address them all and say for sure that one configuration of the variants will certainly lead to a high or low quality system (in terms of some attributes). The uncertainty may also be caused by incomplete information available. Thus we introduce the concept of the *probability* of having a high or low quality system. The probability concept reflects the risks involved in the design decision making [6]. Littlewood and Strigini [27] also answer the questions of why probabilistic reasoning is required by software reliability engineering.

Bayesian Belief Network (BBN, also called Bayesian Network or Probabilistic Networks) is a powerful technique for reasoning under uncertainty [21, 35] and has been successfully applied to a number of domains [13, 14, 17, 18, 22]. In view of the complexity and uncertainty involved in the problem discussed above, we propose a BBN model for the prediction and assessment of the impact of variants on system quality attributes.

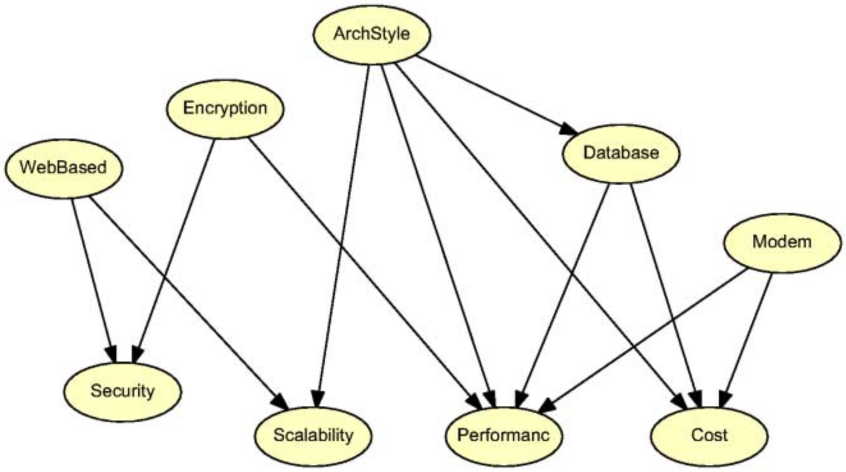
4.1 Bayesian Belief Network – The Graphical Model

Bayesian Belief Network (BBN) is a knowledge representation. It provides a graphical model that resembles human reasoning. A graphical model consists of a set of nodes representing variables and a set of directed edges representing causal/influential relationship among these variables. The variables and directed edges form a directed acyclic graph (DAG).

Figure 3 shows the BBN for the CAD domain. Here, nodes are variables that represent design decisions and quality attributes. A directed edge represents the influential relationship between two nodes (meaning that one has influence on the other). As described in previous section, we have identified eight variants in the CAD product line, out of which the first five (ArchStyle, Database, Web-based, Encryption, Modem) are design decisions to be made by the software developer. The last three variants (Validation, CT-DISP and Dispatch Algorithm) are functional variants that will be determined by the customer and the developer has no control over them, thus we exclude these three variants in the BBN model.

We have identified four quality attributes, namely Performance (High or Low), Security (High or Low), Scalability (High or Low), and Cost (High or Low). It should be noted that the definition of "High" or "Low" depends on specific domain's industrial norm and it varies from domain to domain. For example, for the Performance attribute of one domain, if the industry standard requires that the response time of the target system should not exceed 0.5 ms, then we may define "High" performance as one for which the response time is less than or equal to 0.5 ms, whereas the "Low" performance as one for which the response time is greater than 0.5 ms. However, based on the industrial norm for another domain, the "High" performance may be defined as one for which the response time is less than or equal to 500 ms and the "Low" performance as one for which the response time is greater than 500 ms.

The BBN model (Figure 3) captures domain experts' knowledge and experiences that are accumulated from similar projects in a domain. By studying the BBN, one can see how many design decisions and quality attributes are considered in the CAD architectural design, the design alternatives that each design decision has, and the influential relationships among design decisions and quality attributes. In this aspect, this model resembles the Softgoal Interdependency Graph (SIG) proposed by Chung et al. [9].



Possible Values of Variables:

ArchStyle: Three-Tier, Distributed	Database: Centralized, Distributed
Modem: 9600 bps, 19200 bps	Encryption: Yes, No
WebBased: Yes, No	Security: High, Low
Performance: High, Low	Scalability: High, Low
Cost: High, Low	

Fig. 3. The Bayesian Belief Network for CAD domain (partial)

4.2 Quantifying the Graphical Model

One advantage of the BBN is that it can not only capture the qualitative relationships among variables (denoted by nodes) but also quantify the conceptual relationships. This is done by assigning conditional probability to each node in the BBN. In a BBN, for each variable *v* with parent *Parent*(*v*), there is a corresponding conditional probability distribution *P*(*v*|*Parent*(*v*)) (If *v* has no parents, it has the prior probability *P*(*v*)). For example, in CAD domain, the probability of having high or low scalable system is conditioned by two design decisions: ArchStyle and WebBased. Thus the conditional probability is given as P(Scalability | ArchStyle, WebBased).

A probability number reflects domain expert's belief in how much a given design decision influences quality attributes. These numbers come from either objective data or the experiences of the domain expert accumulated from the development of similar projects. Group-consensus techniques can be used to assess the probabilities from a group of experts. The process of acquiring probabilities from experts is the process of knowledge engineering, in which knowledge engineers conduct interviews or surveys on experts. To assess the probability numbers, one can adapt Boehm's method used in software risk management [6]. In this method, one uses a probability-range estimation to represent the high (0.7 to1.0), medium (0.4 to 0.6) and low (0.0 to 0.3) degree of influence. Value 1.0 is logic True – if P(A=a|B=b) is 1.0, it means that given B=b, A must be a. Value 0.0 is logic False – if P(A=a|B=b) is 0.0, it means that given B=b, A shouldn't be a. Other values between 0.0 and 1.0 are in proportion to the degree of influence.

In our CAD product line, for the Scalability node, we obtain the conditional probability distribution as shown in Table 1.

Table 1. The P (Scalability | ArchStyle, WebBased)

ArchStyle	**WebBased**	**P(Scalability | ArchStyle, WebBased)**	
		High	**Low**
Distributed	Yes	0.9	0.1
Distributed	No	0.6	0.4
Three-Tier	Yes	0.6	0.4
Three-Tier	No	0.3	0.7

The interpretation of these probability numbers is straightforward. For example, when ArchStyle is "Distributed" and when WebBased is "Yes", the probability that the Scalability is "High" is 0.9; whereas when ArchStyle is "Distributed" and when WebBased is "No", the probability that the Scalability is "High" is 0.6; and so on. Again here it should be noted that the definition of "high scalability" is domain-specific.

Similarly, we can assign probabilities to other nodes in Figure 3. Due to space limitation, here we shall not list all the probability tables.

Having quantified the graphical model, we can perform quantitative analysis (e.g., predicting the quality of the target system), which will be described in detail in next section.

5 Configuring Variants in a Product Line

5.1 Reuse of Feature Diagram and BBN Model

Both the feature diagram and the BBN model we have developed so far are assets of an organization – they all capture variant requirements in a domain and can be reused in the reuse-based application engineering.

In reuse-based application engineering for CAD systems, we examine the CAD feature diagram (Figure 2) to understand how many variants are identified in the product line, and to select required variants. This is done by first analyzing customer's requirements for a specific system, and then finding a matching set of features from the feature diagram. It is important that application engineers should work with customers to select the features that the customers really want.

We also examine the BBN model (Figure 3) to understand the impact of design decisions on the quality attributes and to make decisions on adopting or rejecting a design. Not all configurations are "good" with respect to the quality requirements. Different configurations of variants may lead to different systems that satisfy the same set of functional requirements, but differ in certain quality attributes. Very often, we need to make tradeoffs among many "competing" decisions. Thus, during application engineering, we shall carefully configure variants so that the resulting system could satisfy required functional requirements and exhibit desired quality attributes.

It will certainly require extra efforts to construct the feature diagram and the BBN model for a product line. However, these efforts will pay off when the models become organizational assets and are reused for future projects. Based on these assets, one can make rational design decisions and construct target system (a member of the product line) more easily.

5.2 Quality Prediction and Assessment

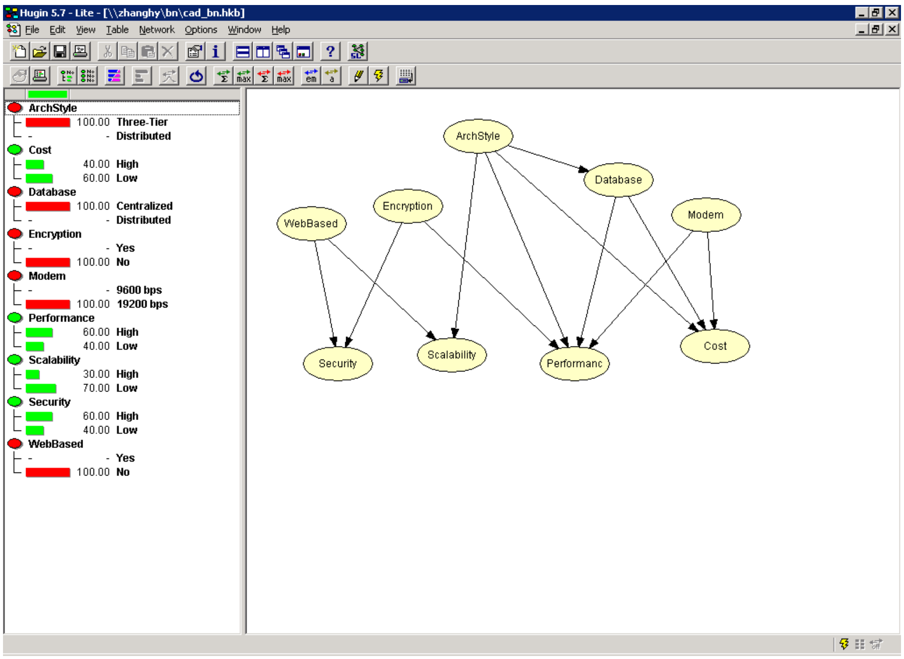
In this section, we illustrate in detail how one can predict the quality of the target system and how rational design decisions can be made based on the analysis of the BBN model.

For example, if an architect designs a specific CAD system as follows (marked as configuration 1):

- ArchStyle = "Three-Tier"
- Database = "Centralized"
- Encryption = "No"
- WebBased = "No"
- Modem = "19200 bps"

We can predict the quality of the resulting CAD system by performing quantitative analysis over the CAD BBN. Assuming the above decisions have made, we can update the BBN in Figure 3 by propagating these decisions over the network and updating the probability distributions of other nodes. There are many inference algorithms available [21, 35]. In our project, we use the *junction tree* method [21], one of the efficient and scalable methods for making inferences in a BBN. The inference can be automated with the aid of BBN tools such as the Hugin tool (available at www.hugin.com). Figure 4 shows the updated BBN constructed by Hugin.

In the left frame of Figure 4, nodes that have a value of 100.00 indicate the decisions made. We can predict that the probability of having high performance is 60%, the probability of having high scalable system is 30%, the probability of having high security is 60%, and the probability of having low cost system is 60%.



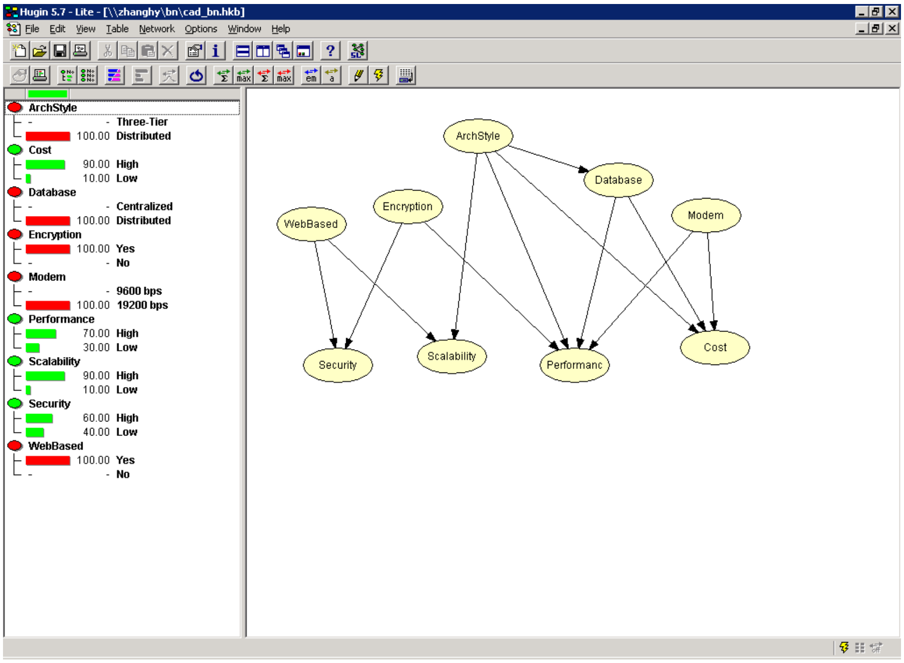
```
ArchStyle
  100.00 Three-Tier
       - Distributed
Cost
   40.00 High
   60.00 Low
Database
  100.00 Centralized
       - Distributed
Encryption
       - Yes
  100.00 No
Modem
       - 9600 bps
  100.00 19200 bps
Performance
   60.00 High
   40.00 Low
Scalability
   30.00 High
   70.00 Low
Security
   60.00 High
   40.00 Low
WebBased
       - Yes
  100.00 No
```

Fig. 4. The updated CAD Bayesian Belief Network (with configuration 1)

For the same CAD system, the architect may also have the following design (marked as configuration 2):

- ArchStyle = "Distributed"
- Database = "Distributed"
- Encryption = "Yes"
- WebBased = "Yes"
- Modem = "19200 bps"

We can predict from Figure 5 that this time, the probability of having high performance system is 70%, the probability of having high scalable system is 90%, the probability of having high secured system is 60%, and the probability of having low cost system is 10%.


Hugin 5.7 - Lite - [\\zhanghy\bn\cad_bn.hkb]
File Edit View Table Network Options Window Help
ArchStyle
Database
Encryption
WebBased
Modem
Security
Scalability
Performanc
Cost

```
ArchStyle
    -       Three-Tier
    100.00  Distributed
Cost
    90.00   High
    10.00   Low
Database
    -       Centralized
    100.00  Distributed
Encryption
    100.00  Yes
    -       No
Modem
    -       9600 bps
    100.00  19200 bps
Performance
    70.00   High
    30.00   Low
Scalability
    90.00   High
    10.00   Low
Security
    60.00   High
    40.00   Low
WebBased
    100.00  Yes
    -       No
```

Fig. 5. The updated CAD Bayesian Belief Network (with configuration 2)

Table 2 shows the comparison between the predicted quality attributes of the target system by configuration 1 and by configuration 2. The numbers shown are the probabilities of having the desired quality attributes. The "+" sign indicates the superior configuration with respect to the quality attribute.

Table 2. The comparison between configuration 1 and configuration 2

	PERFORMANCE (High)	SCALABILITY (High)	SECURITY (High)	COST (Low)
Configuration 1	60%	30%	60%	60% (+)
Configuration 2	70% (+)	90% (+)	60%	10%

Configuration 1 is better than configuration 2 with respect to the system cost. However configuration 1 may lead to a system that is more likely to have low scalability and low performance. Configuration 2 is superior with respect to scalability and performance. However it may result in a system that is more likely to have high cost. For this specific CAD system, if the customer's preference of quality attributes is cost >> security >> performance >> scalability (where ">>" means much more important than), then the configuration 1 is preferable. If the customer's preference is performance >> security >> scalability >> cost, then the configuration 2 is preferable.

To more precisely evaluate a configuration with respects to customer's preference on multiple quality attributes, we can use more rigorous techniques such as the Analytical Hierarchy Process (AHP) [40], which assigns weights to each quality attribute and calculates an overall score to each configuration of design decisions. The better design is the one that has higher overall score.

6 Related Work

Software quality engineering is an important area of software engineering and abundant research has been carried out in this area [16, 23, 31]. Nevertheless, most of the work is focused on quality assurance of a single piece of software product and not many methods are developed to handle the quality issues in a product line. Domain engineering is a process that leads to systematic support for a software product line. To develop a specific software system (a member of the product line), we reuse the assets instead of developing from scratch. Such a systematic, reuse-based approach to software development increases software productivity and quality and has been widely accepted in software engineering practice. However, most of the domain engineering research has been focusing on achieving functional requirements; methods for addressing quality attributes or NFRs are largely informal and ad hoc. Most reusable domain assets are program components, software architectures, and UML models. Domain expert's knowledge and personal experiences in decision making are not well captured and reused.

Chung et al. proposed a non-functional requirement framework [9, 30], in which non-functional requirements are treated as soft-goals (goals that do not have a clear-cut criterion for their satisfaction) and represented graphically as softgoal interdependency graphs (SIGs). SIGs are similar to a Bayesian Belief Network in the way that they can also capture and represent design knowledge.

In last decades Bayesian Belief Network has attracted much attention from both research and industry communities. BBN provides a natural way to structure information about a domain, resembling human reasoning. BBN is based on rigorous graph theory and probability theory, over which quantitative analysis can be conducted. BBN has been successfully applied to many application domains. In software engineering area Fenton and Neil [13, 14] used BBN for software defect prediction and software quality prediction. We use BBN to represent the software design knowledge and experiences for a product line. We can perform both qualitative and quantitative analysis over the constructed BBN to understand how the design decisions influence quality attributes, and to make rational design decisions.

There are also some performance models proposed [3, 37] to establish a link between software performance and software architecture and some adopted the Layered Queuing Network (LQN) approach [38, 44]. However, these performance models do not address other quality attributes such as maintainability or security. Our proposed BBN approach considers many quality attributes and can model the interrelationship among design decisions and quality attributes.

The Architecture Tradeoff Analysis Method (ATAM) [10] and its predecessor Software Architecture Analysis Method (SAAM) [5] evaluate a software architecture mainly after the architecture is designed. These methods require a brainstorming section to generate a list of possible scenarios to test the architectures for single systems. In our approach, we explore design alternatives through careful domain analysis, and produce a BBN that is reusable in a domain.

Experience factory approach [4] defines a framework for experience management, in order to institutionalize the collective learning of an organization. The experience factory approach collects and manages a variety of experience including project data, technology reports and lessons learned, and provides repository services for this experience. As a representation of design knowledge, our BBN could be also stored and managed in an experience factory.

7 Conclusion

In software product line research, methods for handling quality attributes (NFRs) have been largely informal and ad hoc. In this paper, we propose a Bayesian Belief Network based approach to address the problem of analysis and prediction of quality attributes for a product line. We use BBN to capture the design knowledge and experiences of domain experts. Our approach improves the understanding of the impact of design decisions on quality attributes and helps the developer make more informed and rational decisions. For developing specific systems, we reuse this expertise instead of working from scratch.

We believe our approach provides a systematic way of addressing quality attributes for software product lines and helps organization achieve more matured, reuse-based software development. Clearly this research is still at early stage. In the future, we shall evaluate the effectiveness and scalability of our approach in large-scale industrial projects. Better methods and tools shall be developed to facilitate the acquisition of probabilities from experts. For some design decisions and quality attributes, the assessment of probabilities could be improved through the collection of objective data. We shall also integrate the proposed approach with frame-based product line development methods [8, 20, 41]. We hope that our approach could be an integral part of knowledge management practice in a software organization.

References

1. Anderson, B.B., Bajaj, A. and Gorr, W.: An estimation of the decision models of senior IS managers when evaluating the external quality of organizational software, The Journal of Systems and Software, 61 (2002) pp. 59–75
2. Ardis, M., Daley, N., Hoffman, D., Siy, H. and Weiss, D.: Software product lines: a case study, Software-Practice & Experience, 30 (2000) pp. 825–847
3. Aquilani, F., Balsamo, S., Inverardi, P.: Performance analysis at the software architectural design level", Performance Evaluation, 45 (2001) pp. 147–178
4. Basili, V. and Caldiera, G.: Improve software quality by reusing knowledge and experience, *Sloan Management Review*, 37 (1995) pp. 55–64.
5. Bass, L., Clements, P. and Kazman, R.: Software Architecture in Practice, Addison-Wesley (1998)
6. Boehm, B.: Software Risk Management: Principles and Practices, IEEE Software, 8 (1991) pp. 32–41.

7. Boehm, B.W., Brown, J.R., Kaspar, H., Lipow, M., MacLeod, G.J. and Merritt, M.J.: Characteristics of Software Quality. North-Holland Pub. Co., Amsterdam; American Elsevier, New York (1978)
8. Cheong, Y.C. and Jarzabek, S.: Frame-based method for customizing generic software architectures, *Proc. Symposium on Software Reusability*, SSR'99, Los Angeles (1999)
9. Chung, L., Nixon, B.A., Yu, E. and Mylopoulos, J.: *Non-Functional Requirements in Software Engineering*, Kluwer Academic, Boston (2000)
10. Clements, P., Kazman, R. and Klein, M.: *Evaluating Software Architectures – Methods and Case Studies*, Addison-Wesley (2002)
11. Clements, P. and Northrop, L.: *Software Product Lines: Practices and Patterns*. Addison-Wesley (2002)
12. Czarnecki, K. and Eisenecker, U.W.: Generative Programming: Methods, Tools, and Applications. Addison Wesley, Reading, MA, Singapore (2000)
13. Fenton, N. and Neil, M.: A critique of software defect prediction models, IEEE Transactions on Software Engineering, 25 (1999) pp. 675–689.
14. Fenton, N. and Neil, M.: Making decisions: using Bayesian nets and MCDA, Knowledge-Based Systems, 14 (2001) pp. 307–325.
15. Gillies, A.C.: Software Quality: Theory and Management. Chapman & Hall, London (1992)
16. Ginac, F.P. Customer Oriented Software Quality Assurance. Prentice Hall PTR, Upper Saddle River, N.J. (1998)
17. Heckerman, D., Horvitz, E. and Nathwani, B.: Towards normative experts systems: Part I. the Pathfinder project, *Methods of Information in Medicine*, 31 (1992) 90–105.
18. Heckerman, D., Breese, J., and Rommelse, K.: Decision-theoretic troubleshooting. *Communications of the ACM*, 38 (1995) 49-57.
19. ISO/IEC 9126: Information Technology – Software Product Evaluation: Quality Characteristics and Guidelines for their Use. (1991)
20. Jarzabek, S. and Zhang, H.: XML-based method and tool for handling variant requirements in domain models, to appear, *Proc. Fifth IEEE International Symposium on Requirements Engineering (RE'01)*, Toronto, Canada (2001)
21. Jensen, F.V.: *An Introduction to Bayesian Networks*. UCL Press, London (1996)
22. Jensen, F.V., Christensen, H.I., and Nielsen, J.: Bayesian methods for interpretation and control in multi-agent vision systems, *Proc. Applications of Artificial Intelligence*, Orlando (1992)
23. Kan, S.H.: Metrics and Models in Software Quality Engineering. Addison-Wesley, Reading, Mass. (1995)
24. Kang, K. et al.: Feature-Oriented Domain Analysis (FODA) Feasibility Study, Technical Report, CMU/SEI-90-TR-21, Software Engineering Institute, Carnegie Mellon University, Pittsburgh (1990).
25. Kang, K.C.; Lee, J. and Donohoe, P.: Feature-oriented product line engineering, *IEEE Software*, 19 (2002) pp. 58–65.
26. Kitchenham, B.: Software metrics. in Software Reliability Handbook. ed. by P. Rook. Elsevier Applied Science, London, New York (1990)
27. Littlewood B. and Strigini L.: Software reliability and dependability: a roadmap. Proceedings of the conference on The future of Software engineering. International Conference on Software Engineering (ICSE), Limerick, Ireland (2000)
28. Loucopoulos, P. and Karakostas, V.: *System Requirements Engineering*. McGraw-Hill, London, New York (1995)
29. Morisio, M.; Romano, D, and Stamelos, I.: Quality, productivity, and learning in framework-based development: an exploratory case study, IEEE Transactions on Software Engineering, 28 (2002) pp. 876–888.
30. Mylopoulos, J., Chung, L., Liao, S., Wang, H., and Yu, E.: Exploring alternatives during requirements analysis, IEEE Software, 18 (2001) pp. 92–96.
31. Nance, R.E. and Arthur, J.D.: *Managing Software Quality*. Springer, New York (2002).

32. Nixon, B.A.: Management of performance requirements for information systems, IEEE Transactions on Software Engineering, 26 (2000) pp. 1122–1146.
33. O'Regan, G.: *A Practical Approach to Software Quality*, Springer-Verlag, New York (2002)
34. Parnas, D.: On the design and development of program families, *IEEE Transactions on Software Engineering*, 2 (1976) pp. 1–9.
35. Pearl, J.: *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*, Morgan Kaufmann (1988).
36. Perry, W.E.: Effective Methods of EDP Quality Assurance. QED Information Sciences, Wellesley, Mass. (1987)
37. Petriu, D. Shousha, C. and Jalnapurkar, A.: Architecture-based performance analysis applied to a telecommunication system, *IEEE Transactions on Software Engineering*, 26 (2000) pp. 1049–1065
38. Rolia, J.A. and Sevcik, K.C.: The method of layers, *IEEE Transactions on Software Engineering*, 21 (1995) pp. 689–700
39. Roman, G.C.: A taxonomy of current issues in requirements engineering. IEEE Computer, 18 (1985) pp. 14–23
40. Saaty, T.: *The Analytical Hierarchy Process*, McGraw-Hill, New York (1980).
41. Soe, M.S., Zhang, H. and Jarzabek, S.: XVCL: a tutorial, *Proc. of 14th Int. Conf. on Software Engineering and Knowledge Engineering*, SEKE’02, ACM Press, Italy, (2002) pp. 341-349.
42. Weiss, D. and Lai, C.: *Software Product-Line Engineering: A Family-Based Software Development Process*. Addison-Wesley, Reading, MA. (1999)
43. Wieczorek, M. and Meyerhoff, D. (ed.): *Software Quality: State of the Art in Management, Testing, and Tools*. Springer, New York (2001)
44. Woodside, C.M., Neilson, J.E., Petriu, D.C. and Majumdar, S.: The Stochastic Rendezvous Network Model for Performance of Synchronous Client-Server-Like Distributed Software, *IEEE Transactions on Computers*, 44 (1995) pp.20–34