Software Quality Assessment and Project Risk Management Based on Bayesian Belief Networks

Kanaan Abed Faisal
Information and Computer Science Department
King Fahd University of Petroleum and Minerals
Dhahran 31261, Saudi Arabia

Abstract - The ability to reliably predict the risk for a software project presents a significant advantage for a development team. It provides an opportunity to address high risk components earlier in the development life cycle, when their impact is minimized. This article proposes a Bayesian Belief Networks (BBNs) model that captures the evolution of the quality of a software product and provides reliable forecasts of the end quality of the software being developed in light of factors that affect project risk. Insights into risk management, development team skill, software process maturity, and software problem complexity are hypothesized as driving factors of software product quality. The cause-effect relationships between these factors and the elements of software quality and risk management techniques are modeled using BBNs. The BBN software project risk assessment model is utilized to quantify the factors that influence and represent software quality.

Keywords: Bayesian Belief Networks, Reliability, Software Project Risk Assessment.

1 Introduction

Recent surveys suggest that 61% of software projects are delivered over budget, late, or with fewer features originally specified in the scope of the project. There is no doubt that the field of software engineering is fraught with software projects that have been unsuccessful or mismanaged. Inherent uncertainty in software projects has been previously tackled through complex statistical tools and isolated software metrics. Numerous researchers have proposed the use of Bayesian belief networks to cope with this uncertainty [1-12].

The collection of software metrics in isolation is not a reliable method to predict software attributes [1]. Bayesian belief networks are effective primarily because of their nature to allow for the inclusion of all evidence that affect software attributes. We have yet to see research that applies BBNs to software project management holistically.

Software quality is immensely affected by the process with which it is developed. Hence [1-4] propose Bayesian belief networks that aim to improve the software development process. [2,6,7,8,9 10] focus on particular areas of software development and propose models that specifically aim to raise the level of software reliability by predicting possible residual defects. The authors in [5,11] consider factors that affect project risk but again with little consideration of other software quality attributes such as software architecture, defect detection and software quality. In this paper we aim to propose a holistic model that aims to combine the best of the techniques surveyed into a single model that can assist project managers in making a software project successful in all aspects of development; budget, schedule, quality, residual defects, etc.

2 Bayesian Belief Networks

A Bayesian network is a directed acyclic graph associated with a set of probability tables. The graph is made up of nodes representing variables, arcs representing probabilistic dependency relations among the variables and local probability distributions for each variable given values of its parents. Nodes in a BBN represent random variables, whose states are usually expressed in discrete numbers whose states or ranges. Arcs represent the causal relationships between the variables. A Conditional Probability Table (CPT) is associated with each node to denote such causal influence. The node representing a variable A with parent nodes B1, B2, ..., Bn is assigned P(A|B1, B2, ..., Bn) as its CPT. CPTs are filled using a mixture of empirical data and expert judgment, sometimes using the maximum likelihood approach. When root nodes are unknown, current tools usually assign evenly distributed probabilities to these roots. Once new evidence is obtained, evidence can be plugged in the graph followed by re-calculation and updating of nodes to child nodes and vice versa.

A BBN graph can be expanded into an influence diagram by adding decision nodes and utility (cost, or profit) nodes, represented by rectangles and diamonds, respectively. Figure 1 - Sample BBN Topology is a simple example, where the number of defects detected is influenced by problem complexity, the size of the design and the testing efforts applied to the project. BBN’s are criticized for the subjectivity in constructing its influence diagrams and CPT’s. In general, a BBN models the constructor’s belief. Based on this belief, it provides mathematical calculation and prediction. BBNs can be used to support visible and repeatable decision-making. Ziv and Richardson (1997) have used BBN in software testing and maintenance.

Figure 1 - Sample BBN Topology

3 Software Project Risk Assessment

Risk is defined as “The possibility of suffering harm or loss; danger.” Unlike the hazards of daily living, the dangers in the young and emerging field of software engineering must often be learned without the benefit of lifelong exposure [1]. A more deliberate approach is required. Such an approach involves studying the experiences of successful project managers as well as keeping up with the leading writers and thinkers in the field. One such writer in the area of risk is Dr. Barry W. Boehm. In his article “Software Risk Management: Principles and Practices” [2] he lists the following top ten software risk items:

- Personnel Shortfalls
- Unrealistic schedules and budgets
- Developing the wrong functions and properties
- Developing the wrong user interface
- Gold-plating
- Continuing stream of requirements changes
- Shortfalls in externally furnished components
- Shortfalls in externally performed tasks
- Real-time performance shortfalls
- Straining computer-science capabilities

Software Development Life Cycle (SDLC) constitutes the following five steps; initiation, development or accusation, implementation, operation or maintenance and disposal [12]. The above mentioned risks are prevalent through all the phases of SDLC. Project management is the controlling process that governs the execution of the previous mentioned phases. Risk management can be iteratively applied to each phase in the lifecycle to reduce negative impacts on a project.

4 Literature Review

We have conducted an exhaustive survey of research work in the area of BBN assisted software engineering. Various researchers have utilized Belief Networks to assist in reducing costs, risks, time to market and increasing the product quality. The following section summarizes the bulk of the most current research done in this area.

4.1 BBN for Assessing Software Architecture

Software development is done in phases that are usually performed sequentially. Changes or deviations required in later stages compel that the earlier phases be revisited and this impacts the cost, quality and maintainability of the entire system. Therefore this has been an area where a great deal of effort has been done to improve the quality of software, there by reducing the cost and improving the quality at the same time. In the inception phase of a software project only partial and fragmented data is available, in the form of functional and non-functional requirements. This information is used by the designers to create an initial system architecture blue print. Software architecture represents the initial design plan based on which the entire solution or application is developed. Architecture provides the framework and the technical boundary, using which other design and implementation decisions are made.

Table 1 - Software Architecture Assessment Belief-Network Variables

<table>
<thead>
<tr>
<th>Category</th>
<th>Variable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Architecture Attributes</td>
<td>Architecture Style, Class Inheritance depth, Component Granularity, Component Interdependencies, Component Interdependencies, Context Switches, Coupling, Documentation, Dynamic Binding, Exception Handling, Implementation Language, Interface Granularity, Multiple Inheritance, Number of threads</td>
</tr>
<tr>
<td>Quality Criteria</td>
<td>Complexity, Fault Tolerance, Horizontal Complexity, Memory Usage, Responsiveness, Security, Testability, Throughput, Understandability, Vertical Complexity</td>
</tr>
<tr>
<td>Quality Factors</td>
<td>Configurability, Correctness, Flexibility, Maintainability, Modifiability, Performance, Reliability, Reusability, Safety, Scalability, Usability</td>
</tr>
</tbody>
</table>

The quality of software architectures is assessed using qualitative techniques as opposed to quantitative. This research work utilizes Bayesian Belief Networks (BBN) to assess the quality of a system’s architecture, using a combination of qualitative and quantitative data. For the purpose of providing a meaningful support information to make design decisions an initial BBN has been developed called the SAABNet (Software Architecture Assessment Belief Network). Variables within this network have been divided into the following three categories.

The values for the above mentioned variables are two state of yes/no, true/false, high/low etc. apart form Architecture style which has been deemed have four possible values. The BBN based on these variables was
created using Hugin software. This software allows the variables and their relationships to be mapped visually as a BBN. It also allows for probability of variables to be calculated based on the provided evidence. The SABBNet model was used to assess the quality of case studies. Based on the entered evidence regarding architectural attributes SABBNet provided Output for quality attributes given in Table 1 - Software Architecture Assessment Belief-Network Variables. Based on the evaluation of this relatively simplistic network it can be concluded that, it is possible to make qualitative assessment about an architecture that can rival with expert assessments. Meaningful assessment can be made about the quality of software architecture even with limited information available, using Bayesian Belief Networks.

4.2 A Critique Software Defect Prediction

Predicting software quality has always been an important goal for development firms and substantial work has been done on statistical and metric models in this field. Software quality can be predicted based on the presence of defects in a system. These are classified as follows

1. Predicting the number of defects in the system
2. Estimating reliability of system in terms of time to failure
3. Understanding the impact of design and testing processes in defect counts and failure densities.

Defects can be predicted by use of complexity or size metrics but these provide a distorted view. Therefore a different techniques is required that addresses the limitations of statistical and metric based prediction models. Although this paper discusses other techniques of software defect detection, the emphasis in this text is to detail the use of Bayesian Belief Networks, in predicting software defects. The rational behind using BBN for defect prediction is its capability to represent probabilistic relationships among variables. BBNs combine reasoning under uncertainty and the advantages of intuitive visual representation with sound mathematical basis in Bayesians probability. As all defects are not the results of project complexity and are influenced by other factors. Some of these are listed below that explain the presence of defects in a program

- Difficulty of the problem
- Complexity of designed solution
- Programmer/analyst skill
- Design methods and procedures used.

A sample BBN has been constructed with variables that represent major influences in injecting software defects. These variables includes; problem complexity, design effort, design size (KLOC), defects introduced, testing effort, defects detected, defects density at testing, residual defect count and residual defect density. These variables were modeled using the Hugin Explorer tool and provided an evidence data set. As a result the system displays predictions as a histogram based on the entered “facts”. The model propagates these facts and predicts the introduced defects, detected defects and the defect density statistics.

4.3 BBN for Software Suitability Estimation

This paper presents the software quality model which is based on Bayesian Belief Network (BBN) to predict the software quality in the early stages which is essential for the project development team. Predicting the high risk components earlier in the development cycles is useful as their impact can be minimized. This model captures various software quality factors like development team skills, software process maturity, and software problem complexity and provides reliable forecast of the end quality of the software being developed in terms of product suitability. The relationship between these factors and the elements of the software suitability is presented using BBN. Modeling of the software quality involves the current estimation of the development effort and prediction of the quality of the delivered software product. This Software quality model basically provides the projection of the quality of the software product using the software development life cycle.

The software quality model uses the BBN which is a directed acyclic graph consisting of nodes and edges where node represent the random variables and edges represent the relationship between the nodes. Figure 3 shows the high level structure of software quality model using BBN to predict the software quality. Software quality model consist of three inputs and one out. The inputs to the models are rating of development team’s capability, maturity of the project’s development process and the complexity of the software problem. The output of the model is the final software product quality.

Development Team Capability is determined in two ways. First way is to identify the individual team member skills and experience and then assigned some rank (1, 2, 3 or 4). Second way is to provide a method for consolidating the various individual capabilities into a team capability.
**Software Process Maturity** is measured using ISO/IEC 15504, which is comprised of four different process categories that address four distinct areas within a software development project. These areas are project management processes, engineering processes, support processes, and customer supplier processes. **Software Problem Complexity** is captured in requirement, design, implementation, and testing phases differently. **Software Product Quality** uses ISO/IEC 3126, which is an international standard for software product quality. This standard represents the quality of a delivered software product in terms of functionality, efficiency, reliability, usability, maintainability, and portability.

Figure 2 - High-Level Software Quality Model

**Software Quality Model** in Figure 3, relates the inputs and outputs through a set of intermediate nodes, which represent the correctness and completeness of each phase of the development life cycle. The internal structure of the Software Quality Model is depicted in Figure 4. This is basically the Bayesian Network for modeling a quality metric. Each quality metric confirms the correctness and completeness of all the activities associated with each of the four software development life cycle phases. The correctness and completeness will be determined based on software quality drivers, which are development team capability, process maturity, and problem complexity. Figure 5 shows the generic model structure that will be applicable for all these four software development life cycle phases. This model represents how the correctness and completeness can be determined for each software quality indicator which are depicted in Figure 4.

Figure 3 - Bayesian Network for Modeling a Quality Metric

Figure 4 - Bayesian Network for Modeling Phase Correctness and Completeness

### 4.4 Software Reliability: BBN and Fault Trees

For the complex systems like mission critical or safety critical systems, software reliability is a crucial factor. Different design techniques and development methodologies have reduced the complexity of software design and increased the analysis. Software reliability is an important metric to determine overall system reliability. This paper uses Bayesian Belief Network (BBN) and fault trees to estimate the software reliability and presents the dependences between the software engineering processes and software reliability. This proves that BBN and fault trees can improve the software reliability analysis in complex systems.

BBN provide a robust probabilistic framework to evaluate the impact of evidence on uncertain outcomes. BBN network consists of a number of nodes and graphs where nodes represent random variables with some probability distribution values whereas edges represent the connection relationships between the nodes. Fault trees describe the combination of events which cause the system failure. In the hierarchy combination of BBN and fault trees, fault trees are used to compute the probabilities of root nodes for
BBN and BBN are used to compute the probabilities of the basic even in fault trees. Figure 6 shows the Bayesian Belief Network relating processes of software engineering life cycles with the reliability of the software.

The basic idea is to integrate the software standards and development procedures into a BBN to meet the criteria of all the processes in the software engineering cycle. Figure shows the three models which presents the connection of software engineering processes with the initial computed reliability of the software and in the result it generates the new estimated net reliability. If there are not dependencies between the software processes and the software reliability and the overall reliability is equal to initial reliability value and software engineering process is excellent so in that case the reliability will not be more then this and this will be the upper limit but if there is dependencies then the reliability may reduce for any level of the software engineering process.

5 Discussion and Analysis

We have decomposed all aspects of software development into four broad areas and have applied BBN-based techniques in each area with all eventually contributing to the quality of the software product. The following sections are in somewhat of a hierarchy with project risk management encompassing the allocation of resources followed by the lifecycle that focuses on the development process, software reliability on the architecture and design of the system and finally defect detection, that is software quality.

6 Software Project Risk Management

6.1 Initialization

This area pertains with the use of BBNs to minimize software project risk and improve resource allocation. BBNs should continuously assist the decision-making process during the project using feedback so that problematic areas can be dynamically detected and adjusted. The cause-effect relationships in BBNs allow for sources of risks to be identified easily. Also, the ability of BBNs to allow for the introduction of new evidence can improve any iteration to produce estimates of greater reliability. Adapting the risk-management tasks recommended by the IEEE Standard 1540 (2001) and [11], we arrive at the following procedure for the entire software development process:

1. Initialization
2. Maintaining a Project Risk Profile
3. Performing Risk Analysis and Monitoring
4. Performing Risk Treatment

Also, we categorize all the factors that can affect software quality and project risk into two broad groups:

1. Organization-related factors: Factors that involve an organizations risk culture and practice, management experience and capability as well as process maturity.
2. Project-related factors: Factors that involve the nature of the project, complexity, design, architecture, etc
6.2 Organization Related Factors

The BBN in [11] can be modified to include factors mentioned in [5] for greater breadth. Additionally, we need to modify our definition for Design Quality. Design Quality with respect to our goals of generalizing project-related factors will pertain to not only the design of the components, but the overall design of the software system. That is, it shall include the software process lifecycle, the architecture of the different modules as well as the defect related attributes collected as a result of testing.

6.3 Software Project Quality

Most of the research on Bayesian belief networks for software development deals with predicting the residual defects. Residual defects are the bugs that go unnoticed and are discovered after a product is in production. The later these defects are caught, the higher the cost of fixing them. Hence, it is essential for the success of any software project, that defects be caught as early as possible in the software life-cycle, with appropriately allocated resources for testing efforts.

The BBNs in [9,10] in Figure 2, shown the Defect Prediction Model, accurately predicts defect densities and combines evidence from testing data to provide an effective method to predict the probability of defects within the software. The authors in [2] propose the detection of defects in a cyclic fashion, with defects carrying over to each phase of the software project which is inherent in any kind of software development process. This process can be seen in Figure 9 – A Dynamic Bayesian Belief Networks Life Cycle. The BBN and software metrics in [10] combined with the methodology proposed in [2] can result in a defect detection system that is of greater consistency and accuracy.

7 Conclusion

This paper proposed a holistic scheme of developing software with the help of Bayesian belief networks. A comprehensive literature survey in this area was done with a wide-ranging approach to measuring software and project attributes. A BBN-based procedure was adapted to continuously monitor project progress, project risk and assist managers to dynamically adjust resource allocation. The merits of BBNs are they provide visible and repeatable decision-making support under uncertainties. Other BBNs that predicate attributes of software process, reliability and quality were incorporated into our scheme as well. BBNs are advantageous over software metrics due to their ability to consider all diverse factors affecting the problem. Using static analysis, we have shown that the direction of incorporating BBNs in overall software project management and development is promising. Further application of our approach to realistic on-going projects using dynamic experimentation is desired.

Acknowledgement

The Author would like acknowledge the support of King Fahd university of Petroleum and Minerals.

8 Reference


