HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

———————*———————

THESIS

SUBMITTED FOR PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE DEGREE OF

ENGINEER

IN

INFORMATION TECHNOLOGY

SOFTWARE RISK MANAGEMENT USING BAYESIAN NETWORKS

Author:	**Nguyễn Thành Trung**
	Class IS-K56
Supervisor:	**Assoc. Prof.**
	Huỳnh Quyết Thắng

HANOI 05-2017

ACKNOWLEDGEMENTS

I would like to express my deepest gratitude to multitude of people who have taught, helped and supported me during the joyous but also adventurous and challenging time at Hanoi University of Science and Technology.

First and foremost, I would like to sincerely thank my supervisor, Assoc. Prof. Huynh Quyet Thang, who has been constantly giving careful guidance and invaluable advice to help me overcome many challenges during my work. Professor Thang is always dedicated to his students in a way that let them have freedom to express themselves and follow their passions instead of dictating what his students have to do. In addition, he has a strong willingness to spend his time and efforts to connect his students together and form a network so that they can help each other. Working under his supervision, I have learned a lot of invaluable lessons not only in technical knowledge but also in winning friends and influencing people. Therefore I am extremely grateful to have him as my supervisor.

I am also grateful to the current lecturers at School of Information and Communication Technology, for their various forms of support during my college study.

The last seven years is really the hardest time I have ever experienced. Many friends have helped me go through these difficult years. Their supports and care helped me overcome setbacks and stay focused on my study. Thank you for giving me hope and bringing light to my life.

Most importantly, none of this would have been possible without the love and patience of my family. My immediate family to whom this thesis is dedicated to, has been a constant source of love, concern, support and strength after all these years. Without them, I would not have become a mature and responsible person as I am today.

REQUIREMENTS FOR THE THESIS

1. Student information

Student name: Nguyễn Thành Trung

Tel:	0903246964	Email:	trungnguyen.ce@gmail.com
Class:	IS-K56	Program:	HEDSPI

This thesis is performed at: Hanoi University of Science and Technology
From 05/09/2016 to 27/05/2017

2. Goals of thesis

- Study related works about risk management in software development.
- Study theories of probabilistic graphical models, with a focus on Bayesian networks.
- Apply Bayesian networks to risk management in software development problems.

3. Main tasks

- Study related works about risk management in software development.
- Study theories of probabilistic graphical models, with a focus on Bayesian networks.
- Propose a Bayesian network to solve the static risk management problem in software development
- Propose a Bayesian network with better adaptation to the incremental software development process.
- Implement the proposed networks using Python.
- Make experimental evaluations to compare the static model with the incremental model.

4. Declaration of student

I – Nguyen Thanh Trung - hereby warrants that the work and presentation in this thesis are performed by myself under the supervision of Assoc. Prof. Huynh Quyet Thang.

All results presented in this thesis are truthful and are not copied from any other work.

Hanoi, 25/05/2017
Author

Nguyen Thanh Trung

5. Attestation of the supervisor on the fulfillment of the requirements of the thesis

Hanoi, 25/05/2017
Supervisor

Assoc. Prof. Huynh Quyet Thang

TÓM TẮT NỘI DUNG ĐỒ ÁN TỐT NGHIỆP

Ngày nay những dự án phát triển phần mềm luôn bao gồm nguy cơ và rủi ro vì tính chất trừu tượng và đa dạng. Do vậy phân tích, đánh giá và quản lý rủi ro trong công nghệ phần mềm đã trở nên ngày càng quan trọng trong quản lý dự án. Tuy nhiên trong một thời gian dài dự toán rủi ro dự án phần mềm luôn là một vấn đề đau đầu trong công nghệ phần mềm. Đã có rất nhiều phương pháp được đưa ra để giải quyết vấn đề này, nhưng những phương pháp đó chưa giải quyết đầy đủ được tất cả các khía cạnh của vấn đề. Một số mô hình tính toán rủi ro và các phương pháp được đề xuất, dựa trên đánh giá của chuyên gia và ước lượng bằng cách mô phỏng tương tự, nhưng còn nhiều nhược điểm. Thông thường chỉ có các chuyên gia đưa ra ý kiến và nhận định cá nhân để giải quyết vấn đề này, nhưng không giải quyết được vấn đề hoàn toàn vì tính chất đa dạng của các nguy cơ.

Bayesian Network là một lý thuyết rất nổi tiếng với khả năng ứng dụng trong các lĩnh vực công nghệ phần mềm, hỗ trợ chuyên gia đánh giá trong dự toán nguy cơ phần mềm. Và độ chính xác và hiệu quả của lý thuyết này đã được chứng minh trong nhiều dự án thực tế được áp dụng trên thế giới.

Bởi vậy trong đồ án này, tôi đề xuất và phát triển một mô hình ước lượng rủi ro phần mềm được ứng dụng lý thuyết Bayesian Network tập trung tính thực tế của quá trình phát triển phần mềm theo mô hình gia tăng. Tôi đã thực hiện một mô hình thực nghiệm, so sánh kết quả thu được so với mô hình tĩnh và đưa ra kết luận.

Đồ án bao gồm các phần sau :

- Chương 1: Giới thiệu chung bài toán, phương pháp và đóng góp của đồ án
- Chương 2: Lý thuyết Quản lý và ước lượng rủi ro phần mềm
- Chương 3: Tổng quan lý thuyết mạng Bayes
- Chương 4: Mô hình mạng Bayes đề xuất trong bài toán ước lượng rủi ro phần mềm
- Chương 5: Xây dựng chương trình mô phỏng
- Chương 6: Kết quả thí nghiệm và đánh giá
- Chương 7: Kết luận

ABSTRACT OF THESIS

Software development projects offer unique challenges and risks because of their abstract and diverse nature. Risk Analysis and management in software engineering is an emerging part of project management. Software project risk estimation has been a problem in software engineering for many years. A substantial amount of literature has been written to address this issue but this literature does not fully address the concern of the problem. A number of risk estimation models and methods are proposed, based on expert judgment and estimation by analogy, but this area still remains an issue. Normally only expert judgment has been used to address this issue but it does not address the concern of the problem fully because of the diverse nature of risk factors/drivers.

We propose the use of Bayesian Networks, which are famous for their applicability in software engineering areas, to support expert judgment in software risk estimation. Furthermore, in this thesis we propose a Bayesian Network with adaption to the natural incremental characteristic of software development process. We implemented both the static and incremental models, and made experimental evaluations to compare the two models.

This thesis consists of the following sections:

- Chapter 1: Introduction: problem, objective and project contributions
- Chapter 2: Risk management in Software depvelopment
- Chapter 3: An overview of Bayesian networks
- Chapter 4: Proposed Bayesian network models in Software development risk management problem
- Chapter 5: Model implementation
- Chapter 6: Experimental results and evaluations
- Chapter 7: Conclusion

Contents

List of Figures

Part I

Introduction

Chapter 1

Project Introduction

1.1 Motivation

Many risks are involved in software development and risk management has become one of the key activities in software development. Bayesian networks (BNs) have been explored as a tool for various risk management practices, including the risk management of software development projects as software project failures are often a result of insufficient and ineffective risk management. However, much of the present research on software risk analysis focuses on finding the correlation between risk factors and project outcome in a static context. To obtain proper and effective risk control, risk planning should be performed based on a dynamic context to provide more risk information for decision making.

In this study, we propose a model using BNs for risk analysis of software development projects in a dynamic context which describe the accordingly time-based software development process (i.e. incremental model).

1.2 Contributions of the thesis

Following the stated objective, the contributions of my thesis are:

1. *Propose a novel Bayesian network model for Software risk development estimation.*

   To the best of my knowledge, currently there is no Bayesian model that considers software development projects in a dynamic context. We propose a model which targets to describe the accordingly time-based software development process. This is also the main objective of my thesis.

2. *Implement proposed model and do experimental evaluations to compare it with current model.*

In this thesis, both the static and dynamic models are implemented using Python. Then the proposed model is compared with current existing model to evaluate its performance and is proved to make significant improvement in eliminating risks in the software development process.

3. *Evaluate the proposed model*

   We proposed some evaluation metrics and scenarios for experiments. The experiments were performed to compare the proposed dynamic model with the current static model. The results shown that our proposed model is better than the other in all evaluation metrics.

1.3 Organization of thesis

The remainder of this thesis is organized as follows. In chapter 2, we discuss about the Risk management in Software development. In chapter 3, we give an overview of the Bayesian Networks. The application of Bayesian Networks to Risk estimation is presented in chapter 4 where we proposed different models of Bayesian Networks in both static and dynamic context. The source code and detailed simulation of the models is provided in chapter 5. In chapter 6, we present the experimental results to evaluate the performance of the models. Finally, we conclude the thesis in chapter 7.

Part II

Theoretical Background

Chapter 2

Risk management in Software depvelopment

2.1 Introduction

2.1.1 Risk

Risk is the possibility of loss. It is a function of both the probability of an adverse event occurring and its impact; the impact manifests itself in a combination of financial loss, time delay, and loss of performance. A risk is the precursor to a problem; the probability that, at any given point in the software life cycle, the predicted goals cannot be achieved within available resources. Risk can not be eliminated from a software project, but it can be managed. Risk management is critical to the success of any software effort and is a strategic aspect of all software projects.

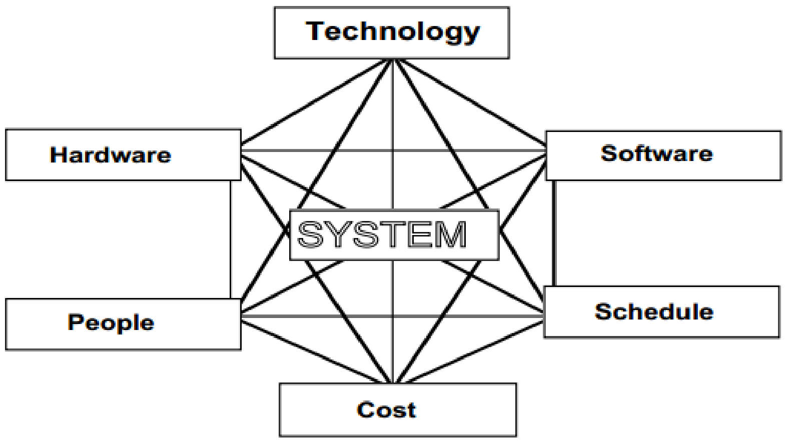


Figure 2.1: Risks Within a System Context

2.1.2 Software risk management

Software risk management is a software engineering practice with processes, methods, and tools for managing risks in a project. It provides a disciplined environment for proactive decision-making to assess continuously what can go wrong; determine what risks are important to deal with; and implement actions to deal with those risks. Risk management planning addresses the strategy for risk management, the risk management process, and the techniques, methods, and tools to be used to support the risk management process.

A software project may encounter various types of risks:

- Technical risks include problems with languages, project size, project functionality, platforms, methods, standards, or processes. These risks may result from excessive constraints, lack of experience, poorly defined parameters, or dependencies on organizations outside the direct control of the project team.
- Management risks include lack of planning, lack of management experience and training, communications problems, organizational issues, lack of authority, and control problems.
- Financial risks include cash flow, capital and budgetary issues, and return on investment constraints.
- Contractual and legal risks include changing requirements, market-driven schedules, health & safety issues, government regulation, and product warranty issues.
- Personnel risks include staffing lags, experience and training problems, ethical and moral issues, staff conflicts, and productivity issues.
- Other resource risks include unavailability or late delivery of equipment & supplies, inadequate tools, inadequate facilities, distributed locations, unavailability of comp uter resources, and slow response times.

The need to manage risk increases with system complexity. Figure 1.2 demonstrates this concept by indicating that as the complexity of the system increases, both technical and non-technical (cost and schedule) risks increase. There is an increasing need for more systematic methods and tools to supplement individual knowledge, judgment, and experience. These human traits are often sufficient to address less complex risks. It is worth noting that many managers believe that they are managing risk in its multifaceted dimensions. The fact of the matter is that they are merely managing cost and schedule along with isolated cases of technical risk.

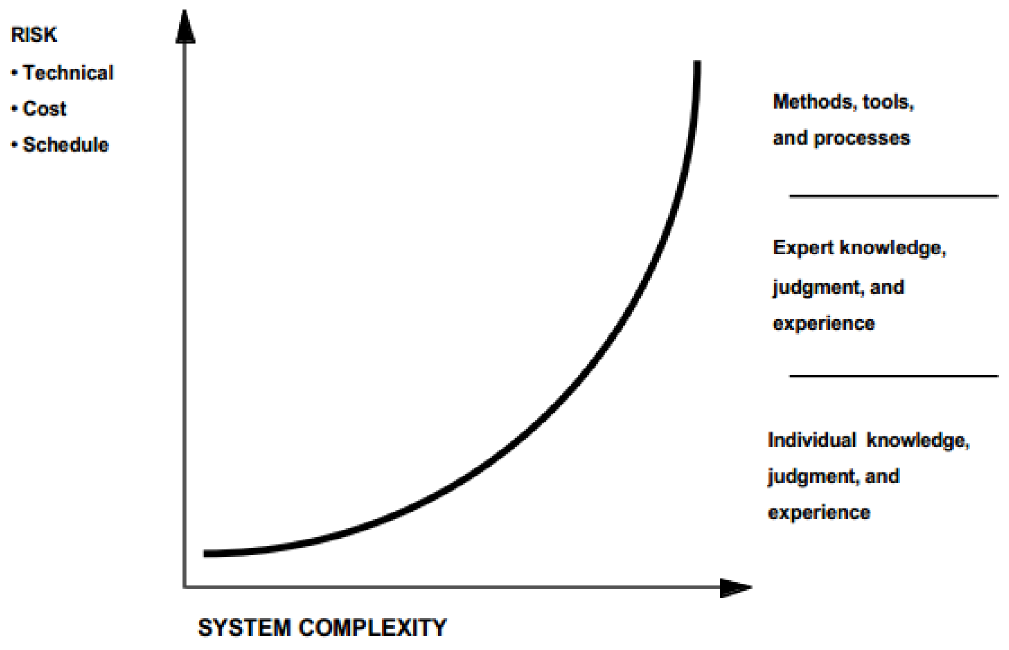


Figure 2.2: The Need to Manage Risk Increases With System Complexity

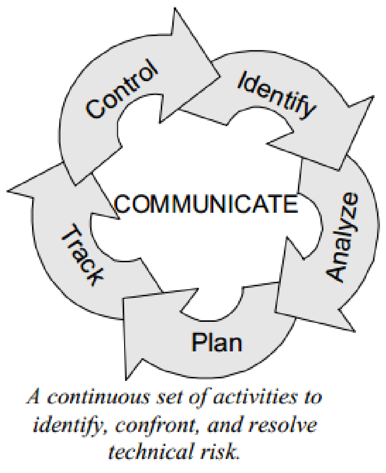
2.1.3 Role of management

Senior management support and commitment is critical to the success of any risk management initiative. A formal risk management process requires corporate acceptance of risk as a major consideration for software management. Senior management must support project risk management activities by: (1) providing adequate personnel, budget, schedules, and other resources (e.g., tools and equipment); (2) ensuring project management receives the required training in identifying, managing, and communicating software risks; and (3) ensuring project personnel receive the required training in conducting risk management tasks. Senior management reviews risk management activities on a periodic and event-driven basis.

2.2 Risk management process

2.2.1 Process model

Figure 1.3 illustrates the risk management process developed by the Software Engineering Institute, Carnegie Mellon University[1]. This process starts with the identification of a list of potential risks. Each of these risks is then analyzed and prioritized. A risk management plan is created that identifies containment actions that will reduce the probability of the risk occurring and/or reduce the impact if the risk turns into a problem. The plan also includes contingency actions that will be taken if the risk turns into a problem and the associated triggers (indicators that the risk is turning into a problem). The containment part of the plan is then implemented and actions are taken. The tracking step involves monitoring the status of known risks as well as the results of risk reduction actions. If a trigger indicates the onset of a problem, the corresponding contingency plans are implemented. As new status and information are obtained, the risk management plans are updated accordingly. Tracking may also result in the addition of newly identified risks or in the closure of known risks.


Control
Identify
COMMUNICATE
Analyze
Track
Plan
A continuous set of activities to identify, confront, and resolve technical risk.

Identify	Search for and locate risks before they become problems adversely affecting the project
Analyze	Process risk data into decision-making information
Plan	Translate risk information into decisions and actions (both present and future) and implement those actions
Track	Monitor the risk indicators and actions taken against risks
Control	Correct for deviations from planned risk actions
Communicate	Provide visibility and feedback data internal and external to your program on current and emerging risk activities

Figure 2.3: SEI Software Risk Management Model

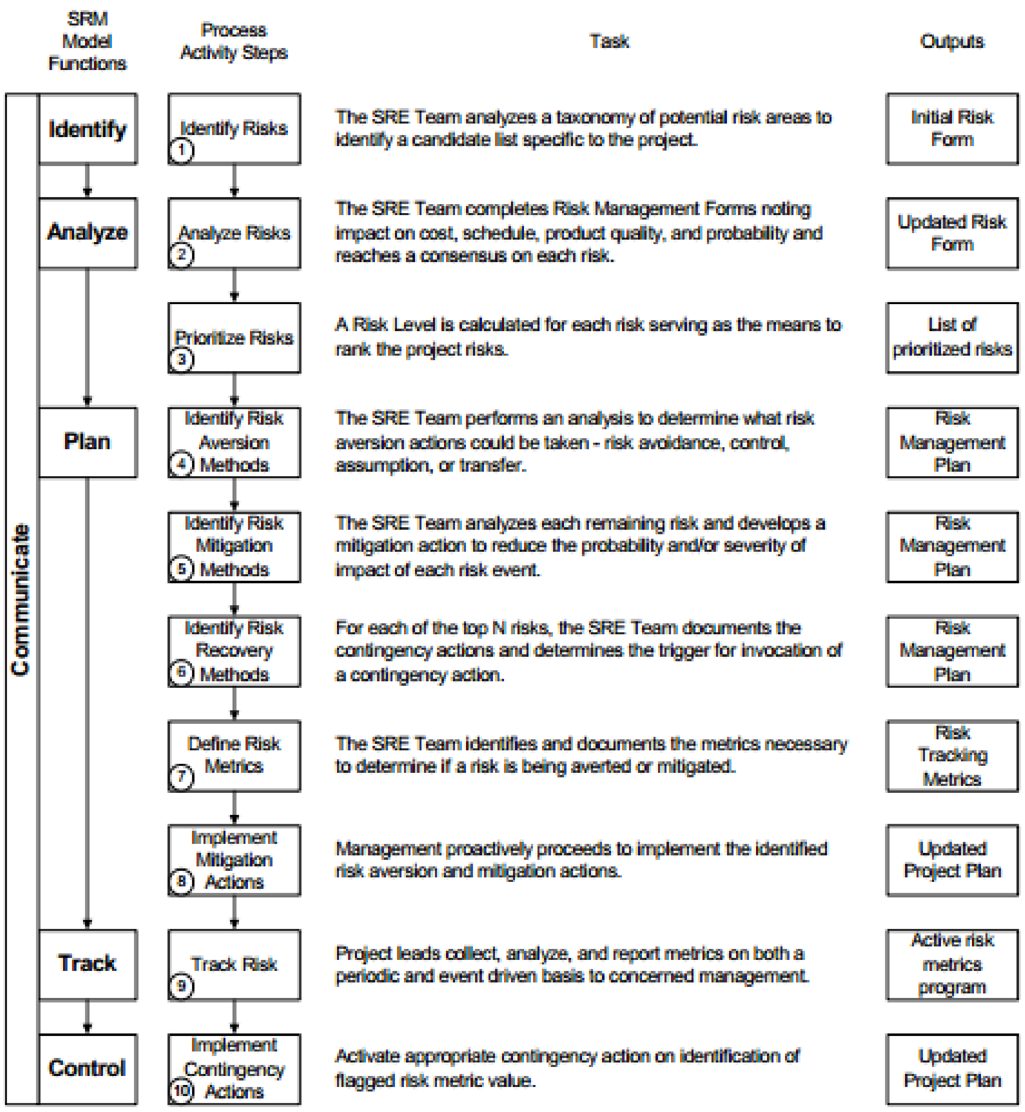
Many of the most serious issues encountered in system acquisition are the result of risks that either remain unrecognized and/or are ignored until they have already created serious consequences. This focus on risk management is important because structured techniques, even quite simple ones, can be effective in identifying risk, and approaches, procedures, and techniques do exist for risk mitigation.

The risk management process is an on-going part of managing the software development process. It is designed to be a continuous feedback loop where additional information and risk status are utilized to refine the project's risk list and risk management plans.

2.2.2 Process procedure

An overview of the risk management process is illustrated in Figure 1.4. This process description is based on the software risk management process defined by the Software Engineering Process Office and the Software Engineering Institute.

The risk management process consists of ten steps as described in the paragraphs that follow. Use of the activities associated with these steps constitutes an acceptable risk management approach and could be incorporated into a Risk Management Plan. The size, visibility, or consequences of the project drives the complexity of the process. The process can be tailored to be consistent with existing site project management processes.



SRM Model Functions	Process Activity Steps	Task	Outputs
Identify	1 Identify Risks	The SRE Team analyzes a taxonomy of potential risk areas to identify a candidate list specific to the project.	Initial Risk Form
Analyze	2 Analyze Risks	The SRE Team completes Risk Management Forms noting impact on cost, schedule, product quality, and probability and reaches a consensus on each risk.	Updated Risk Form
	3 Prioritize Risks	A Risk Level is calculated for each risk serving as the means to rank the project risks.	List of prioritized risks
Plan	4 Identify Risk Aversion Methods	The SRE Team performs an analysis to determine what risk aversion actions could be taken - risk avoidance, control, assumption, or transfer.	Risk Management Plan
	5 Identify Risk Mitigation Methods	The SRE Team analyzes each remaining risk and develops a mitigation action to reduce the probability and/or severity of impact of each risk event.	Risk Management Plan
	6 Identify Risk Recovery Methods	For each of the top N risks, the SRE Team documents the contingency actions and determines the trigger for invocation of a contingency action.	Risk Management Plan
	7 Define Risk Metrics	The SRE Team identifies and documents the metrics necessary to determine if a risk is being averted or mitigated.	Risk Tracking Metrics
	8 Implement Mitigation Actions	Management proactively proceeds to implement the identified risk aversion and mitigation actions.	Updated Project Plan
Track	9 Track Risk	Project leads collect, analyze, and report metrics on both a periodic and event driven basis to concerned management.	Active risk metrics program
Control	10 Implement Contingency Actions	Activate appropriate contingency action on identification of flagged risk metric value.	Updated Project Plan

Figure 2.4: Software Risk Management Process Overview

1. **Function 1: Identify**

   Before risks can be managed, they must be identified, and they must be identified before they become problems adversely affecting the project. Establishing an environment that encourages people to raise concerns and issues and conducting quality reviews throughout all phases of a project are common techniques for identifying risks.

   During the first step in the software risk management process, risks are identified and added to the list of known risks. The output of this step is a list of project-specific risks that have the potential of compromising the project's success. There are many techniques for identifying risks, including interviewing, reporting, decomposition, assumption analysis, critical path analysis, and utilization of risk taxonomies.

   Interviewing/Brainstorming: One technique for identifying risks is interviewing or brainstorming with project personnel, customers, and vendors. Open-ended questions such as the following can help identify potential areas of risk.

- What new or improved technologies does this project implement?
- What interfaces issues still need to be defined?
- What requirements exist that we aren't sure how to implement?
- What concerns do we have about our ability to meet the required quality and performance levels?

Voluntary Reporting: Another risk identification technique is voluntary reporting, where any individual who identifies a risk is encouraged and rewarded for bringing that risk to management's attention. This requires the complete elimination of the "shoot the messenger" syndrome. It avoids the temptation to assign risk reduction actions to the person who identified the risk. Risks can also be identified through required reporting mechanisms such as status reports or project reviews.

Decomposition: As the product is being decomposed during the requirements and design phases, another opportunity exists for risk identifications. Every TBD ("To Be Done/Determined") is a potential risk. As Ould states, "The most important thing about planning is writing down what you don't know, because what you don't know is what you must find out". Decomposition in the form of work breakdown structures during project planning can also help identify areas of uncertainty that may need to be recorded as risks.

Assumption Analysis: Process and product assumptions must be analyzed. For example, we might assume the hardware would be available by the system test date or three additional experienced C++ programmers will be hired by the time coding starts. If these assumptions prove to be false, we could have major problems. Critical Path Analysis: As we perform critical path analysis for our project plan, we must remain on the alert to identify risks. Any possibility of schedule slippage on the critical path must be considered a risk because it directly impacts our ability to meet schedule.

Risk Taxonomies: Risk taxonomies are lists of problems that have occurred on other projects and can be used as checklists to help ensure all potential risks have been considered. An example of a risk taxonomy can be found in the Software Engineering Institute's Taxonomy -Based Risk Identification report that covers 13 major risk areas with about 200 questions.

2. **Function 2: Analyze**

   Analysis is the conversion of risk data into risk decision-making information. It includes reviewing, prioritizing, and selecting the most critical risks to address.

During the risk analysis step, each risk is assessed to determine:

- Likelihood: the probability that the risk will result in a loss
- Impact: the size or cost of that loss if the risk turns into a problem
- Timeframe: when the risk needs to be addressed (i.e., risk associated with activities in the near future would have a higher priority then similar risks in later activities)

Additionally, the interrelationships between risks are assessed to determine if compounding risk conditions magnify losses.

The project and risk management personnel evaluating the Risk Level for each risk can determine when appropriate mitigation action will be required. This decision making can be facilitated by the use of risk levels where the Risk Levels are defined as:

(a) Tolerable Risk is a condition where risk is identified as having little or no effect or consequence on project objectives; the probability of occurrence is low enough to cause little or no concern.

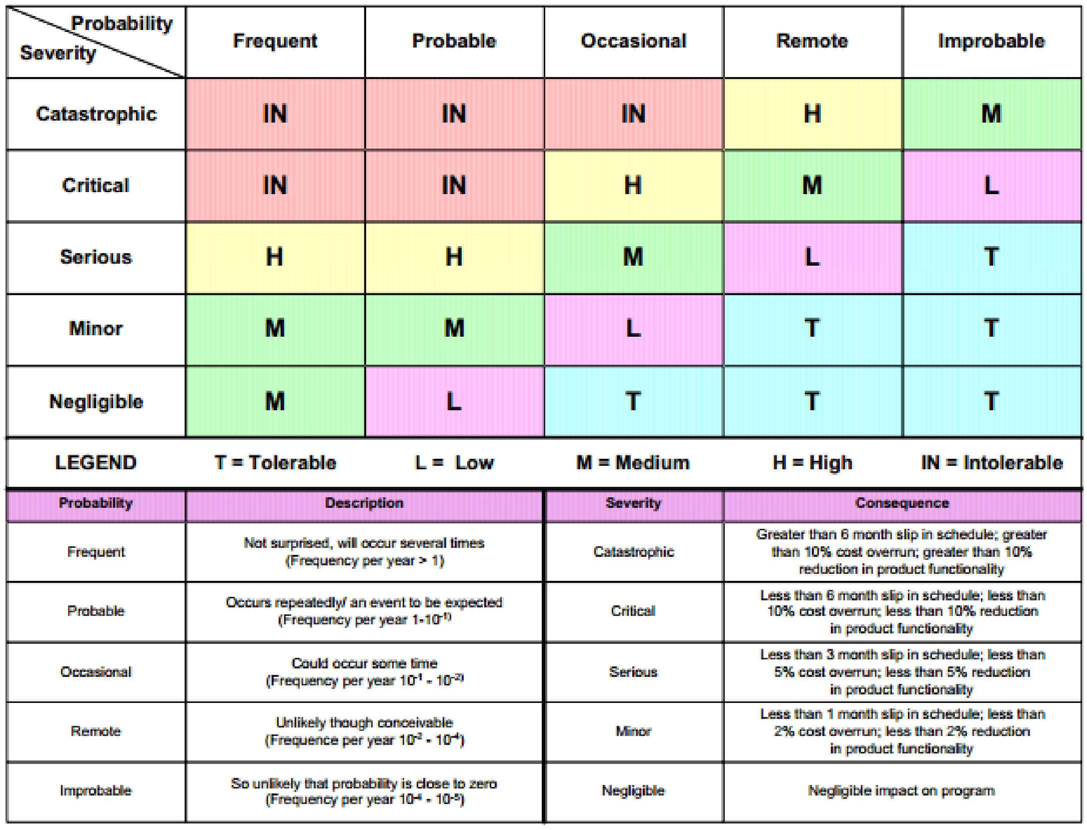
(b) Low Risk is a condition where risk is identified as having minor effects on project objectives; the probability of occurrence is sufficiently low to cause only minor concern.

(c) Medium Risk is a condition where risk is identified as one that could possibly affect project objectives, cost, or schedule. The probability of occurrence is high enough to require close control of all contributing factors.

(d) High Risk is the condition where risk is identified as having a high probability of occurrence and the consequence would affect project objectives, cost, and schedule. The probability of occurrence is high enough to require close control of all contributing factors, the establishment of risk actions, and an acceptable fallback position.

(e) Intolerable Risk is the condition where risk is identified as having a high probability of occurrence and the consequence would have significant impact on cost, schedule, and/or performance. These risks would constitute the Top N for the project. At the conclusion of risk prioritization, a consolidated list of risks is created, and the updated Risk Management Forms are placed under configuration management.

The following is an example of risk analysis. During our analysis, we determine that there is a 30% probability the Test Bed will be available one week later than scheduled and a 10% probability it will be a month late. If the Test Bed is one week late, the testers can use their time productively by using the simulators to



Severity \ Probability	Frequent	Probable	Occasional	Remote	Improbable
Catastrophic	IN	IN	IN	H	M
Critical	IN	IN	H	M	L
Serious	H	H	M	L	T
Minor	M	M	L	T	T
Negligible	M	L	T	T	T
LEGEND	T = Tolerable	L = Low	M = Medium	H = High	IN = Intolerable

Probability	Description	Severity	Consequence
Frequent	Not surprised, will occur several times (Frequency per year > 1)	Catastrophic	Greater than 6 month slip in schedule; greater than 10% cost overrun; greater than 10% reduction in product functionality
Probable	Occurs repeatedly/ an event to be expected (Frequency per year 1-10^{-1})	Critical	Less than 6 month slip in schedule; less than 10% cost overrun; less than 10% reduction in product functionality
Occasional	Could occur some time (Frequency per year 10^{-1} - 10^{-2})	Serious	Less than 3 month slip in schedule; less than 5% cost overrun; less than 5% reduction in product functionality
Remote	Unlikely though conceivable (Frequence per year 10^{-2} - 10^{-4})	Minor	Less than 1 month slip in schedule; less than 2% cost overrun; less than 2% reduction in product functionality
Improbable	So unlikely that probability is close to zero (Frequency per year 10^{-4} - 10^{-5})	Negligible	Negligible impact on program

Figure 2.5: Sample Risk Matrix

test other aspects of the software (loss = \$0). The simulator can be utilized for up to two weeks. However, if the Test Bed delivery is one month late, there are not enough productive activities to balance the loss. Losses include unproductive testers for two weeks, overtime later, morale problems, and delays in finding defects for a total estimated loss of \$100,000. In addition to the dollar loss, the testing is on the critical path and not all of the lost testing time can be made up in overtime (loss estimated at two week schedule slippage).

Boehm defines the Risk Exposure equation to help quantitatively establish risk priorities. Risk Exposure measures the impact of a risk in terms of the expected value of the loss. Risk Exposure (RE) is defined as the probability of an undesired outcome times the expected loss if that outcome occurs.

$$\text{RE} = \text{Probability(UO)} * \text{Loss (UO)}$$

where UO = Unexpected outcome.

Given the example above, the Risk Exposure is 10% x \$100,000 = \$10,000 and \$10% x 2 calendar week = 0.2 calendar week. Comparing the Risk Exposure measurement for various risks can help identify those risks with the greatest probable negative impact to the project or product and thus help establish which risks are candidates for further action.

The list of risks is then prioritized based on the results of our risk analysis. Since resource limitations rarely allow the consideration of all risks, the prioritized list of risks is used to identify risks requiring additional planning and action.

Other risks are documented and tracked for possible future consideration. Based on changing conditions, additional information, the identification of new risks, or the closure of existing risks, the list of risks requiring additional planning and action may require periodic updates.

3. **Function 3: Plan**

   Taking the prioritized risk list as input, plans are developed for the risks chosen for action. Figure 2 illustrates the specific questions that can be asked to help focus on the type of planning required.

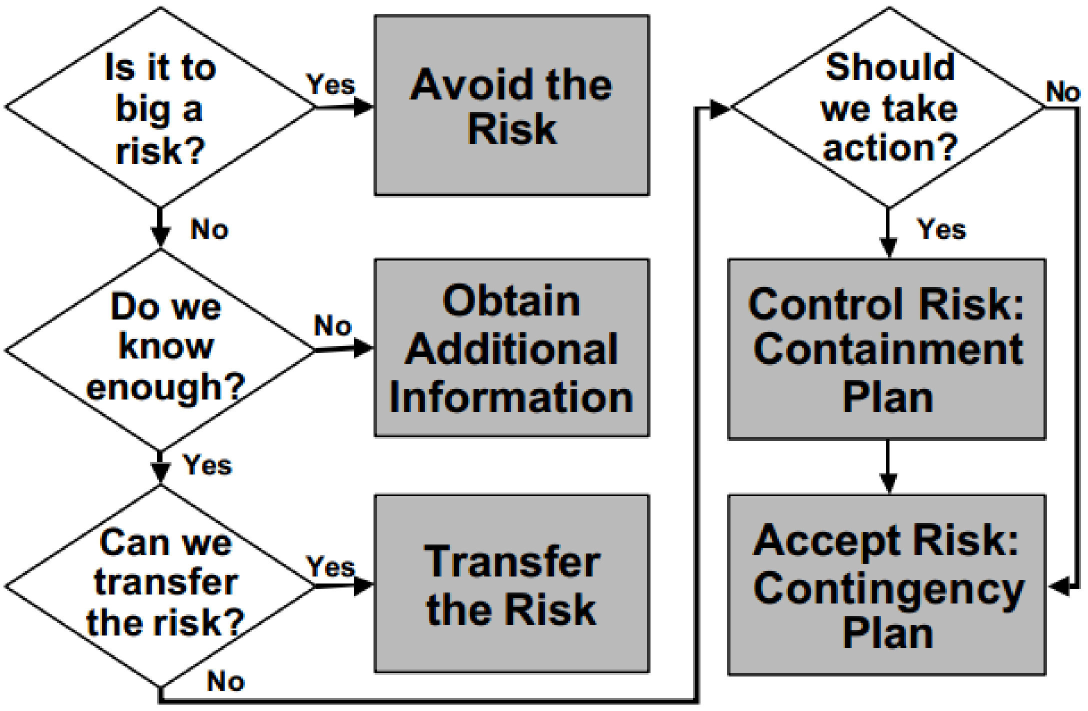

Is it to big a risk?
Yes
Avoid the Risk
No
Do we know enough?
No
Obtain Additional Information
Yes
Can we transfer the risk?
Yes
Transfer the Risk
No
Should we take action?
No
Yes
Control Risk: Containment Plan
Accept Risk: Contingency Plan

Figure 2.6: Techniques for Handling Risks

We will use the following two risks to illustrate the types of actions that might be taken using each risk handling technique:

- The subcontractor may not deliver the software at the required reliability level and as a result the reliability of the total system may not meet performance specifications.
- The interface with the new control device is not defined and as a result its driver may take more time to implement then scheduled.

Is it too big a risk?

If the risk is too big for us to be willing to accept, we can avoid the risk by changing our project strategies and tactics to choose a less risky alternate or we may decide not to do the project at all. For example, if our project has tight schedule constraints and includes state of the art technology, we may decide to wait until a future project to implement our newly purchased CASE tools.

Things to remember about avoiding risks include:

- Avoiding risks may also mean avoiding opportunities
- Not all risks can be avoided
- Avoiding a risk in one part of the project may create risks in other parts of the project

Do we know enough?

If we don't know enough, we can plan to "buy" additional information through mechanisms such as prototyping, modeling, simulation, or conducting additional research.

Can we transfer the risk?

If it is not our risk or if it is economically feasible to pay someone else to assume all or part of the risk, we can plan to transfer the risk to another organization. For example we can contract with a disaster recovery firm to provide backup computer facilities that will allow continuation of the project in case a fire or other disaster destroys the project's work environment.

Should action be taken now?

If we decide to attack the risk directly, we typically start with creating a list of possible risk reduction actions that can be taken for the risk. Two major types of risk reduction actions should be considered:

(a) Actions that reduce the likelihood that the risk will occur

(b) Actions that reduce the impact of the risk should it occur

These may include actions such as establishing a liaison with the customer to insure adequate communications, conducting a performance simulation, or buying additional equipment for the test bed to duplicate the operational environment.

From the list of possible risk reduction actions, we must select those that we are actually going to implement. When considering which risk reduction activities to select, a cost/benefit analysis should be performed. Boehm defines the Risk Reduction Leverage equation to help quantitatively establish the cost/benefit of implementing a risk reduction action. [Boehm-89]. Risk Reduction Leverage measures the return on investment of the available risk reduction techniques. Risk Reduction Leverage (RRL) is defined as the difference between the Risk Exposure before and after the reduction activity divided by the cost of that activity.

RRL = (REbefore - REafter) / Risk Reduction Cost

If the Risk Reduction Leverage is less than one, it means that the cost of the risk reduction activity outweighs the probable gain from implementing the action.

Contingency plans:

If immediate risk reduction actions are not taken or if those actions reduce but do not eliminate the risk, it may be appropriate to develop risk contingency plans. Contingency plans are plans that are implemented only if the risk actually turns into a problem. Examples of contingency plans include disaster recovery plans, contacting a consulting firm to establish a fallback position if key personnel are not available, or selecting an alternative design approach if the new technology is not delivered as promised.

Each risk that has a contingency plan should also have a trigger. A trigger is a time or event in the future that is the earliest indication that the risk will turn into a problem. For example, if there is a risk that outsourced software will not be delivered on schedule, the trigger could be whether the critical design review was held on schedule. A trigger can also be a relative variance or threshold metric. For example, if the risk is the availability of key personnel for the coding phase, the trigger could be a relative variance of more than 10% between actual and planned staffing levels.

By default, all of the risks that didn't have a high enough priority to warrant action planning are risks that we have assumed. We should also consider assigning triggers to these risks as indicators that detect status or priority changes. For example, because the probability of both the test bed and simulator hardware not being delivered in time for System Test is so small, we did not select this risk for action. However, we may want to set a trigger at the Integration Test Readiness Review to reexamine that risk.

Risks that are assigned triggers can be set at a "monitor only" status until the trigger occurs. At that time, the risk analysis step should be repeated to determine if risk reduction action is needed or if the contingency plan should be implemented.

There are trade-offs in utilizing triggers in risk management. We want to set the trigger as early as possible in order to ensure that there is plenty of time to implement risk reduction actions. We also want to set the trigger as late as possible because the longer we wait, the more information we have to make a correct decision and not implement unnecessary actions.

Adjusting the project plan: Each selected action in the risk handling plans must include a description of the action and a list of tasks with assigned responsibilities and due dates. These actions must be integrated into the project plan with effort and cost estimations. Project schedules must be adjusted to include these new actions. For example, new tasks such as creating prototypes, doing research or conducting alpha testing at the customer's site must be included in the project plan.

4. **Function 4: Taking Action**

   During the action step, we implement the risk reduction plan. Individuals execute their assigned tasks. Project effort estimations are adjusted to take into consideration additional effort needed to perform risk reduction activities or to account for projected additional effort if the risk turns into a problem. Some tasks may be moved forward in the schedule to ensure adequate time to deal with problems if they occur. Other tasks may be moved back in the schedule to allow time for additional information to be obtained. Budgets must also be adjusted to consider risk reduction activities.

5. **Function 5: Tracking**

   Results and impacts of the risk reduction implementation must be tracked. The tracking step involves gathering data, compiling that data into information, and then reporting and analyzing that information. This includes measuring known risks and monitoring triggers, as well as measuring the impacts of risk reduction activities. The results of the tracking can be:

   - Identification of new risks that need to be added to the risk list.
   - Validation of known risk resolutions so risks can be removed from the risk list because they are no longer a threat to project success.
   - Information that dictates additional planning requirements
   - Implementation of contingency plan

   Many of the software metrics typically used to manage software projects can also be used to track risks. For example, Gantt charts, earned value measures, and budget and resource metrics can help identify and track risks involving variances between plans and actual performance. Requirements churn, defect identification rates, and defect backlogs can be used to track rework risks, risks to the quality of the delivered product, and even schedule risks.

Chapter 3

An overview of Bayesian networks

3.1 Introduction

Bayesian networks (BNs) are graphical models for reasoning under uncertainty, where the nodes represent variables (discrete or continuous) and arcs represent direct connections between them. These direct connections are often causal connections. In addition, BNs model the quantitative strength of the connections between variables, allowing probabilistic beliefs about them to be updated automatically as new information becomes available.

In this chapter we will describe how Bayesian networks are put together (the syntax) and how to interpret the information encoded in a network (the semantics). We will look at how to model a problem with a Bayesian network and the types of reasoning that can be performed.

3.2 Bayes' Theorem

In probability theory and statistics, Bayes' theorem (alternatively Bayes' law or Bayes' rule) describes the probability of an event, based on prior knowledge of conditions that might be related to the event. For example, if cancer is related to age, then, using Bayes' theorem, a person's age can be used to more accurately assess the probability that they have cancer, compared to the assessment of the probability of cancer made without knowledge of the person's age.

Bayes' Theorem:

Let the m events $B_1, B_2, ..., B_m$ constitute a partition of the sample space S. That is, the B_i are mutually exclusive:

$$B_i \cap B_j = \emptyset \text{ for } i \neq j$$

and exhaustive:

$$S = B_1 \cup B_2 \cup ... \cup B_m$$

Also, suppose the prior probability of the event B_i is positive, that is, $P(B_i) > 0$ for $i = 1, ..., m$. Now, if A is an event, then A can be written as the union of m mutually exclusive events, namely:

$$A = (A \cap B1) \cup (A \cap B_2) \cup ... \cup (A \cap B_m)$$

Therefore:

$$\begin{aligned} P(A) &= P(A \cap B_1) + P(A \cap B_2) + \ldots + P(A \cap B_m) \\ &= \sum_{i=1}^{m} P(A \cup B_i) \\ &= \sum_{i=1}^{m} P(B_i) \times P(A|B_i) \end{aligned}$$

And so, as long as $P(A) > 0$, the posterior probability of event B_k given event A has occurred is:

$$\begin{aligned} P(B_k|A) &= \frac{P(B_k \cap A)}{P(A)} \\ &= \frac{P(B_k) \times P(A|B_k)}{\sum_{i=1}^{m} P(B_i) \times P(A|B_i)} \end{aligned}$$

3.3 Bayesian network representation

A Bayesian network is a graphical structure that allows us to represent and reason about an uncertain domain. The nodes in a Bayesian network represent a set of random variables from the domain. A set of directed arcs (or links) connects pairs of nodes, representing the direct dependencies between variables. Assuming discrete variables, the strength of the relationship between variables is quantified by conditional probability distributions associated with each node. The only constraint on the arcs allowed in a BN is that there must not be any directed cycles: you cannot return to a node simply by following directed arcs. Such networks are called directed acyclic graphs, or simply dags.

Throughout the remainder of this section we will use the following simple medical diagnosis problem.

Example problem: Lung cancer

A patient has been suffering from shortness of breath (called dyspnoea) and visits the doctor, worried that he has lung cancer. The doctor knows that other diseases, such as tuberculosis and bronchitis, are possible causes, as well as lung cancer. She also *knows that other relevant information includes whether or not the patient is a smoker (increasing the chances of cancer and bronchitis) and what sort of air pollution he has been exposed to. A positive X-ray would indicate either TB or lung cancer.*

3.3.1 Nodes and values

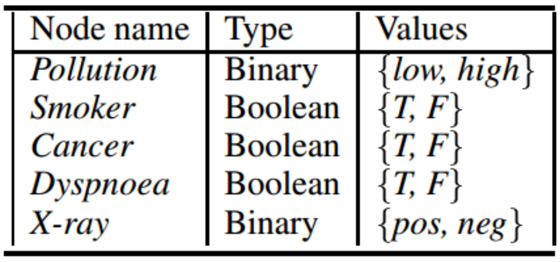
First, the knowledge engineer must identify the variables of interest. This involves answering the question: what are the nodes to represent and what values can they take? For now we will consider only nodes that take discrete values. The values should be both mutually exclusive and exhaustive, which means that the variable must take on exactly one of these values at a time. Common types of discrete nodes include:

1. Boolean nodes, which represent propositions, taking the binary values true (T) and false (F). In a medical diagnosis domain, the node Cancer would represent the proposition that a patient has cancer.

2. Ordered values. For example, a node Pollution might represent a patient's pollution exposure and take the values *low, medium, high.*

3. Integral values. For example, a node called Age might represent a patient's age and have possible values from 1 to 120.

Even at this early stage, modeling choices are being made. For example, an alternative to representing a patient's exact age might be to clump patients into different age groups, such as *baby, child, adolescent, young, middle-aged, old*. The trick is to choose values that represent the domain efficiently, but with enough detail to perform the reasoning required.



Node name	Type	Values
Pollution	Binary	{*low, high*}
Smoker	Boolean	{*T, F*}
Cancer	Boolean	{*T, F*}
Dyspnoea	Boolean	{*T, F*}
X-ray	Binary	{*pos, neg*}

Figure 3.1: Preliminary choices of nodes and values for the lung cancer examples

3.3.2 Structure

The structure, or topology, of the network should capture qualitative relationships between variables. In particular, two nodes should be connected directly if one affects or causes the other, with the arc indicating the direction of the effect. So, in our medical diagnosis example, we might ask what factors affect a patient's chance of having cancer?

If the answer is “Pollution and smoking,” then we should add arcs from Pollution and Smoker to Cancer. Similarly, having cancer will affect the patient’s breathing and the chances of having a positive X-ray result. So we add arcs from Cancer to Dyspnoea. The resultant structure is shown in Figure 3.2. It is important to note that this is just one possible structure for the problem.

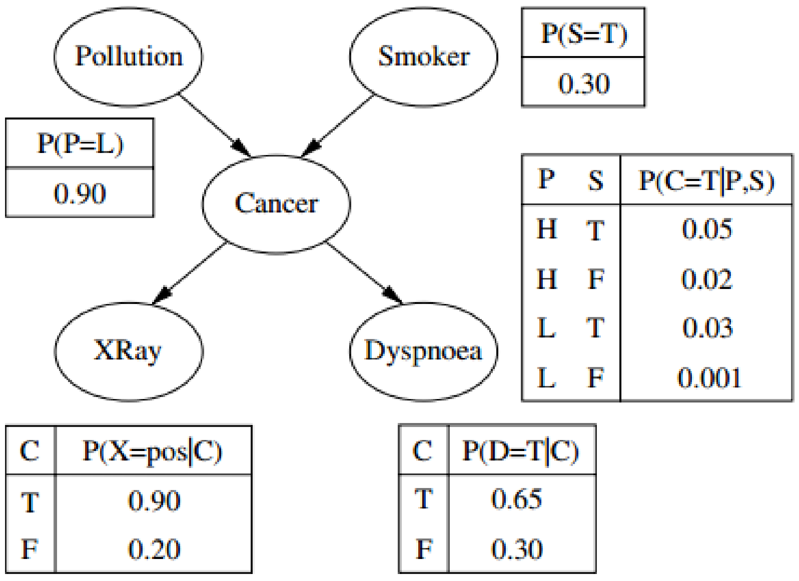


Figure 3.2: A BN for the lung cancer problem

Structure terminology and layout In talking about network structure it is useful to employ a family metaphor: a node is a **parent** of a **child**, if there is an arc from the former to the latter. Extending the metaphor, if there is a directed chain of nodes, one node is an **ancestor** of another if it appears earlier in the chain, whereas a node is a **descendant** of another node if it comes later in the chain. In our example, the Cancer node has two parents, Pollution and Smoker, while Smoker is an ancestor of both X-ray and Dyspnoea. Similarly, X-ray is a child of Cancer and descendant of Smoker and Pollution. The set of parent nodes of a node X is given by *Parents(X)*.

Another useful concept is that of the **Markov blanket** of a node, which consists of the node’s parents, its children, and its children’s parents. Other terminology commonly used comes from the “tree” analogy (even though Bayesian networks in general are graphs rather than trees): any node without parents is called a **root** node, while any node without children is called a **leaf** node. Any other node (non-leaf and non-root) is called an intermediate node. Given a causal understanding of the BN structure, this means that root nodes represent original causes, while leaf nodes represent final effects. In our cancer example, the causes Pollution and Smoker are root nodes, while the effects X-ray and Dyspnoea are leaf nodes.

By convention, for easier visual examination of BN structure, networks are usually laid out so that the arcs generally point from top to bottom.

3.3.3 Conditional probabilities

Once the topology of the BN is specified, the next step is to quantify the relationships between connected nodes – this is done by specifying a conditional probability distribution for each node. As we are only considering discrete variables at this stage, this takes the form of a conditional probability table (CPT).

First, for each node we need to look at all the possible combinations of values of those parent nodes. Each such combination is called an instantiation of the parent set. For each distinct instantiation of parent node values, we need to specify the probability that the child will take each of its values.

For example, consider the Cancer node of Figure 3.2. Its parents are Pollution and Smoking and take the possible joint values $< H, T >, < L, T >, < H, F >, < L, F >$. The conditional probability table specifies in order the probability ofcancer for each of these cases to be:0.05, 0.02, 0.03, 0.001. Since these are probabilities, and must sum to one over all possible states of the Cancer variable, the probability of no cancer is already implicitly given as one minus the above probabilities in each case; i.e., the probability of no cancer in the four possible parent instantiations is 0.95, 0.98, 0.97, 0.999

Root nodes also have an associated CPT, although it is degenerate, containing only one row representing its prior probabilities. In our example, the prior for a patient being a smoker is given as 0.3, indicating that 30% of the population that the doctor sees are smokers, while 90% of the population are exposed to only low levels of pollution.

Clearly, if a node has many parents or if the parents can take a large number of values, the CPT can get very large. The size of the CPT is, in fact, exponential in the number of parents. Thus, for Boolean networks a variable with n parents requires a CPT with 2^{n+1} probabilities.

3.3.4 The Markov property

2.2.4 The Markov property In general, modeling with Bayesian networks requires the assumption of the **Markov property**: *there are no direct dependencies in the system being modeled which are not already explicitly shown via arcs*. In our Cancer case, for example, there is no way for smoking to influence dyspnoea except by way of causing cancer (or not) — there is no hidden “backdoor” from smoking to dyspnoea. Bayesian networks which have the Markov property are also called **Independence-maps** (or, **I-maps** for short), since every independence suggested by the lack of an arc is real in the system.

Whereas the independencies suggested by a lack of arcs are generally required to exist in the system being modeled, it is not generally required that the arcs in a BN correspond to real dependencies in the system. The CPTs may be parameterized in such a way as to nullify any dependence. Thus, for example, every fully-connected Bayesian network can represent, perhaps in a wasteful fashion, any joint probability distribution over the variables being modeled. Of course, we shall prefer minimal models and, in particular, minimal I-maps, which are I-maps such that the deletion of any arc violates I-mapness by implying a non-existent independence in the system.

If, in fact, every arc in a BN happens to correspond to a direct dependence in the system, then the BN is said to be a **Dependence-map** (or, **D-map** for short). A BN which is both an I-map and a D-map is said to be a **perfect map**.

3.4 Understanding Bayesian networks

We now consider how to interpret the information encoded in a BN — the probabilistic **semantics** of Bayesian networks.

3.4.1 Representing the joint probability distribution

Most commonly, BNs are considered to be representations of joint probability distributions. There is a fundamental assumption that there is a useful underlying structure to the problem being modeled that can be captured with a BN, i.e., that not every node is connected to every other node. If such domain structure exists, a BN gives a more compact representation than simply describing the probability of every joint instantiation of all variables. Sparse Bayesian networks (those with relatively few arcs, which means few parents for each node) represent probability distributions in a computationally tractable way.

Consider a BN containing the n nodes, X_1 to X_n, taken in that order. A particular value in the joint distribution is represented by $P(X_1 = x_1, X_2 = x_2, ..., X_n = x_n)$, or more compactly, $P(x_1, x_2, ..., x_n)$. The **chain rule** of probability theory allows us to factorize joint probabilities so:

$$\begin{aligned} P(x_1, x_2, ..., x_n) &= P(x_1) \times P(x_2|x_1)..., \times P(x_n|x_1, ..., x_{n-1}) \\ &= \prod_i P(x_i|x_1, ..., x_{i-1}) \end{aligned}$$

Recalling that the structure of a BN implies that the value of a particular node is conditional only on the values of its parent nodes, this reduces to

$$P(x_1, x_2, ..., x_n) = \prod_i P(x_i|Parents(X_i))$$

3.4.2 Causal chains

Bayesian networks which satisfy the Markov property (and so are I-maps) explicitly express conditional independencies in probability distributions. The relation between conditional independence and Bayesian network structure is important for understanding how BNs work.

Consider a causal chain of three nodes, where A causes B which in turn causes C, as shown in Figure 3.4(a). In our medical diagnosis example, one such causal chain is "smoking causes cancer which causes dyspnoea." Causal chains give rise to conditional independence, such as for Figure 3.4(a):

$$P(C|A \wedge B) = P(C|B)$$

This means that the probability of C, given B, is exactly the same as the probability of C, given both B and A. Knowing that A has occurred doesn't make any difference to our beliefs about C if we already know that B has occurred. We also write this conditional independence as: $A \perp C|B$

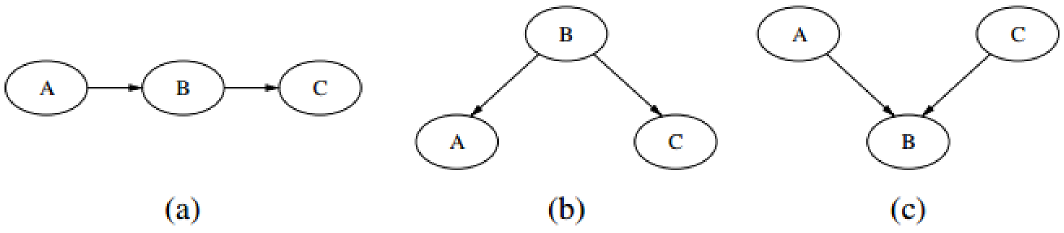


Figure 3.3: (a) Causal chain; (b) common cause; (c) common effect

3.4.3 Common causes

Two variables A and C having a common cause B is represented in Figure 3.3(b).

In our example, cancer is a common cause of the two symptoms, a positive X-ray result and dyspnoea. Common causes (or common ancestors) give rise to the same conditional independence structure as chains:

$$P(C|A \wedge B) = P(C|B) \equiv A \perp C|B$$

If there is no evidence or information about cancer, then learning that one symptom is present will increase the chances of cancer which in turn will increase the probability of the other symptom. However, if we already know about cancer, then an additional positive X-ray won't tell us anything new about the chances of dyspnoea.

3.4.4 Common Effects

A common effect is represented by a network v-structure, as in Figure 3.3(c). This represents the situation where a node (the effect) has two causes. Common effects (or their descendants) produce the exact opposite conditional independence structure to that of chains and common causes. That is, the parents are marginally independent $A \perp C$, but become dependent given information about the common effect (i.e., they are conditionally dependent):

$$P(C|A \wedge B) = P(A|C) \equiv \neg(A \perp C|B)$$

Thus, if we observe the effect (e.g., cancer), and then, say, we find out that one of the causes is absent (e.g., the patient does not smoke), this raises the probability of the other cause (e.g., that he lives in a polluted area) — which is just the inverse of explaining away

3.5 Reasoning with Bayesian networks

Now that we know how a domain and its uncertainty may be represented in a Bayesian network, we will look at how to use the Bayesian network to reason about the domain. In particular, when we observe the value of some variable, we would like to condition upon the new information. The process of conditioning (also called **probability propagation** or **inference** or **belief updating**) is performed via a "flow of information" through the network. Note that this information flow is not limited to the directions of the arcs. In our probabilistic system, this becomes the task of computing the posterior probability distribution for a set of **query** nodes, given values for some **evidence** (or **observation**) nodes.

3.5.1 Types of reasoning

Bayesian networks provide full representations of probability distributions over their variables. That implies that they can be conditioned upon any subset of their variables, supporting any direction of reasoning.

For example, one can perform **diagnostic reasoning**, i.e., reasoning from symptoms to cause, such as when a doctor observes Dyspnoea and then updates his belief about Cancer and whether the patient is a Smoker. Note that this reasoning occurs in the opposite direction to the network arcs.

Or again, one can perform **predictive reasoning**, reasoning from new information about causes to new beliefs about effects, following the directions of the network arcs. For example, the patient may tell his physician that he is a smoker; even before any symptoms have been assessed, the physician knows this will increase the chances of the patient having cancer. It will also change the physician's expectations that the patient will exhibit other symptoms, such as shortness of breath or having a positive X-ray result.

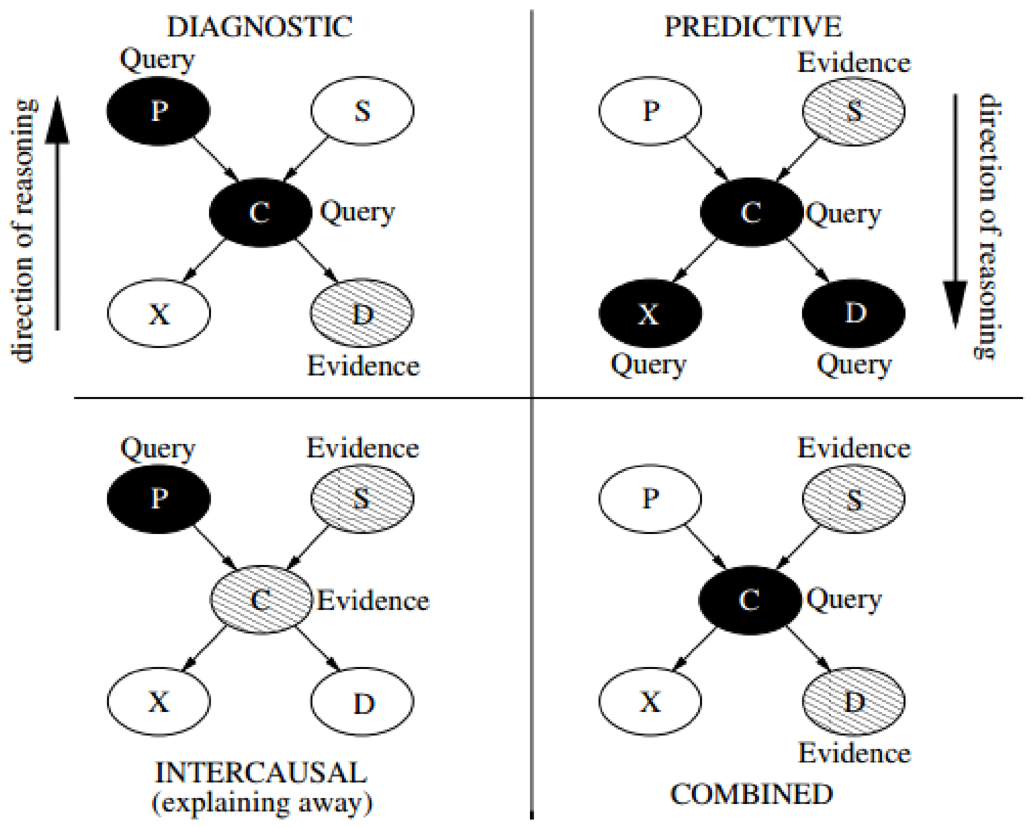


Figure 3.4: Types of reasoning

3.5.2 Types of evidence

So Bayesian networks can be used for calculating new beliefs when new information – which we have been calling **evidence** – is available. In our examples to date, we have considered evidence as a definite finding that a node X has a particular value, x, which we write as $X = x$. This is sometimes referred to as **specific evidence**.

For example, suppose we discover the patient is a smoker, then $Smoker = T$, which is specific evidence.

However, sometimes evidence is available that is not so definite. The evidence might be that a node Y has the value y_1 or y_2 (implying that all other values are impossible). Or the evidence might be that T is not in state y_1 (but may take any of its other values); this is sometimes called a **negative evidence**. In fact, the new information might simply be any new probability distribution over Y . Suppose, for example, that the radiologist who has taken and analyzed the X-ray in our cancer example is uncertain. He thinks that the X-ray looks positive, but is only 80% sure. In Bayesian networks this is also known as virtual evidence. Since it is handled via likelihood information, it is also known as likelihood evidence.

Part III

Implementation

Chapter 4

Proposing Bayesian network models in the Software risk estimation problem

4.1 Model for Static software development process

The developers of any new complex software system will confirm that, no matter how much testing they perform, there will still be plenty of defects (or ‘bugs’) yet to be found. The hope is that, when the software is released, any defects found by end-users will have minimal impact. Hence, the decision about when to stop testing and release the software must always be balanced by the likely number (and criticality) of remaining defects. It follows that the ability to produce accurate predictions of ‘residual’ defects in software systems is one of the most important and challenging tasks confronting software engineers. It is especially relevant for safety critical software (such as in transport and medical systems where software that is released with too many defects can have life-threatening impact); but the business of any commercial software producer can be devastated if they get their release decision wrong.

In 2006 Prof. Norman Fenton wrote a paper about software defect prediction that was published in the IEEE Transactions on Software Enginering. The paper was a critique of the commonly used approaches to the problem and they proposed a novel approach based on causal models. Although the 1999 paper provided no validation of the new approach, it became one of the most cited papers in computer science. According to Essential Science Indicators the paper is in the top 1% most influential papers.

In the the paper, Fenton proposed a causal model for software defect prediction using BNs as follows:

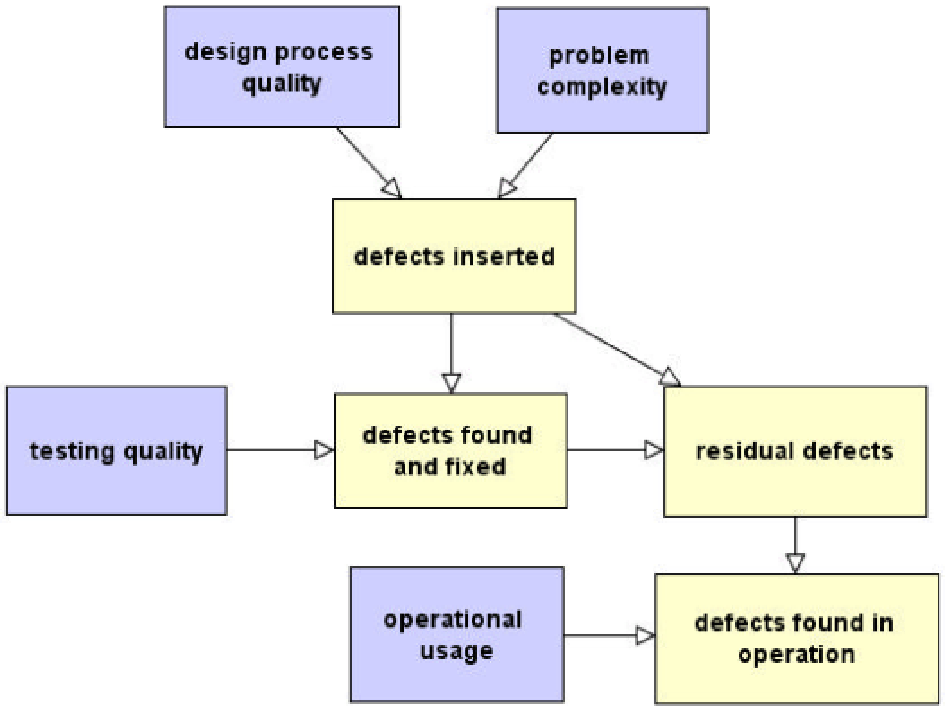


Figure 4.1: Fenton's causal model for software defect prediction

In the model the number of operational defects (those found by customers) in a software module is what we are really interested in predicting. We know this is clearly dependent on the number of residual defects. But it is also critically dependent on the amount of operational usage. If you do not use the system you will find no defects irrespective of the number there (the model allows you to predict this with perfect precision). The number of residual defects is determined by the number you introduce during development minus the number you successfully find and fix. Obviously defects found and fixed is dependent on the number introduced.

The number introduced is influenced by problem complexity and design process quality. The better the design the fewer the defects and the less complex the problem the fewer defects. Finally, how many defects you find is influenced not just by the number there to find but also by the amount of testing effort.

4.2 Model for Dynamic software development process

The model by Fenton, however, shows a limitation in term that it does not consider the . Meanwhile in reality, software development is a continous process (i.e. in incremental software development). Therefore, we propose a new model adding temporal factor with three time frames as follows:

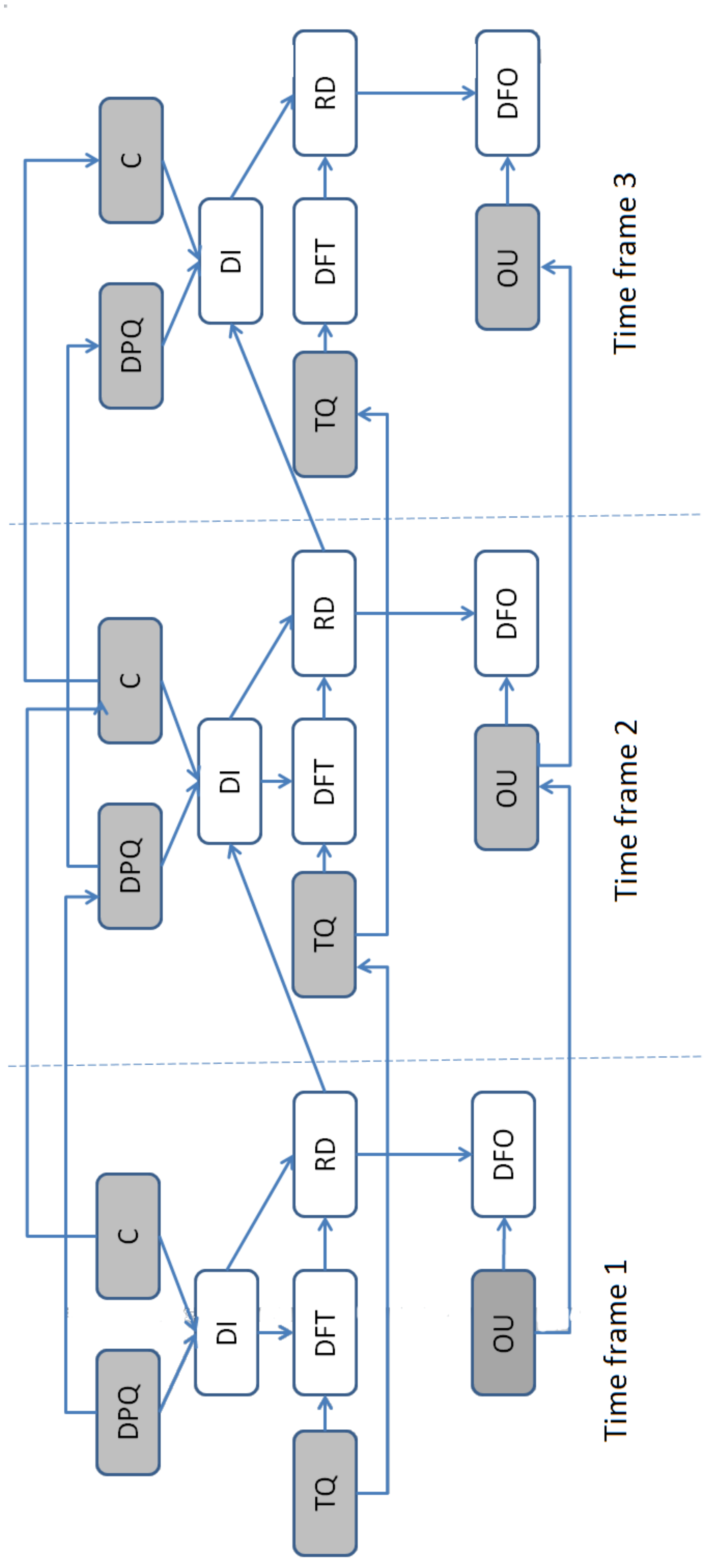


Figure 4.2: Our proposed model for software defect prediction

In this scenario, logically the nodes Design Process Quality (DPQ), Complexity (C), Testing Quality (TQ), Operational Usage (OU) holds the same values in every time stamp. This is due to the fact that this model assumes that the model uses multiple Testing steps so Design Process Quality, Complexity, Operational Usage should stay unchanged.

While Testing Quality may vary, it is assumed that the same tester works for all the time stamps. For the same reason, Operational Usage is supposed to be unchanged.

The number of Defect Inserted after each step of testing is set to be the Residual Defects of the previous time frame.

Chapter 5

Model Implementation

5.1 Environment, Language and Libraries

The model was implemented using the following Environment, Language and Libraries:

1. **Environment:**
    - Operating System: Windows 8.1
    - CPU: Intel Core i5 (dual core)
    - RAM: 8GB
2. **Environment:** Python 2.7

    Python is a widely used high-level programming language for general-purpose programming. An interpreted language, Python has a design philosophy which emphasizes code readability (notably using whitespace indentation to delimit code blocks rather than curly brackets or keywords), and a syntax which allows programmers to express concepts in fewer lines of code than possible in languages such as C++ or Java. The language provides constructs intended to enable writing clear programs on both a small and large scale.

    Python features a dynamic type system and automatic memory management and supports multiple programming paradigms, including object-oriented, imperative, functional programming, and procedural styles. It has a large and comprehensive standard library.

    Python interpreters are available for many operating systems, allowing Python code to run on a wide variety of systems.
3. **Libraries:**
    - **pgmpy**

pgmpy is a python library for working with graphical models. It allows the user to create their own graphical models and answer inference or map queries over them. pgmpy has implementation of many inference algorithms like VariableElimination, Belief Propagation etc.

- **NumPy**

  NumPy is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays.

  NumPy targets the CPython reference implementation of Python, which is a non-optimizing bytecode interpreter. Mathematical algorithms written for this version of Python often run much slower than compiled equivalents. NumPy address the slowness problem partly by providing multidimensional arrays and functions and operators that operate efficiently on arrays, requiring (re)writing some code, mostly inner loops using NumPy.

  Using NumPy in Python gives functionality comparable to MATLAB since they are both interpreted, and they both allow the user to write fast programs as long as most operations work on arrays or matrices instead of scalars. In comparison, MATLAB boasts a large number of additional toolboxes, notably Simulink, whereas NumPy is intrinsically integrated with Python, a more modern and complete programming language. Moreover, complementary Python packages are available; SciPy is a library that adds more MATLAB-like functionality and Matplotlib is a plotting package that provides MATLAB-like plotting functionality. Internally, both MATLAB and NumPy rely on BLAS and LAPACK for efficient linear algebra computations.

- **pandas**

  pandas is a software library written for the Python programming language for data manipulation and analysis. In particular, it offers data structures and operations for manipulating numerical tables and time series.

- **Matplotlib**

  Matplotlib is a Python 2D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments across platforms. Matplotlib can be used in Python scripts, the Python and IPython shell, the jupyter notebook, web application servers, and four graphical user interface toolkits.

  pyplot is a matplotlib module which provides a MATLAB-like interface. matplotlib is designed to be as usable as MATLAB, with the ability to use Python, with the advantage that it is free.

- **Tkinter**

  Tkinter is a Python binding to the Tk GUI toolkit. It is the standard Python interface to the Tk GUI toolkit, and is Python's de facto standard GUI.

  As with most other modern Tk bindings, Tkinter is implemented as a Python wrapper around a complete Tcl interpreter embedded in the Python interpreter. Tkinter calls are translated into Tcl commands which are fed to this embedded interpreter, thus making it possible to mix Python and Tcl in a single application.

5.2 Program structure

The simulation consists of the following modules:

1. **Generating data** Since getting the data of software development risks and defects is not feasible for its secretcy, time-consumming and only-available-to-big-software-corporation characteristics, in the scope of this thesis, we have no choice but to use a set of generated data.
2. **Processing data**

   This module reads the csv file - output from the Data Generating module. It also

The details of each module are discussed in the following section

5.3 Input and Output

5.3.1 Input

5.3.2 Output

The output of the program, are the conditional probabilities of all the nodes.

5.4 Detailed module designs

Chapter 6

Experimental results and evaluations

This section gives the details of all experiments that we have conducted to explore and evaluate different models as well as the effects of different parameters to the models proposed in previous chapter. When comparing a certain hyperparameter, all the other ones are fixed within an experiment.

6.1 Experiments with the Static model

6.1.1 Worst case scenario

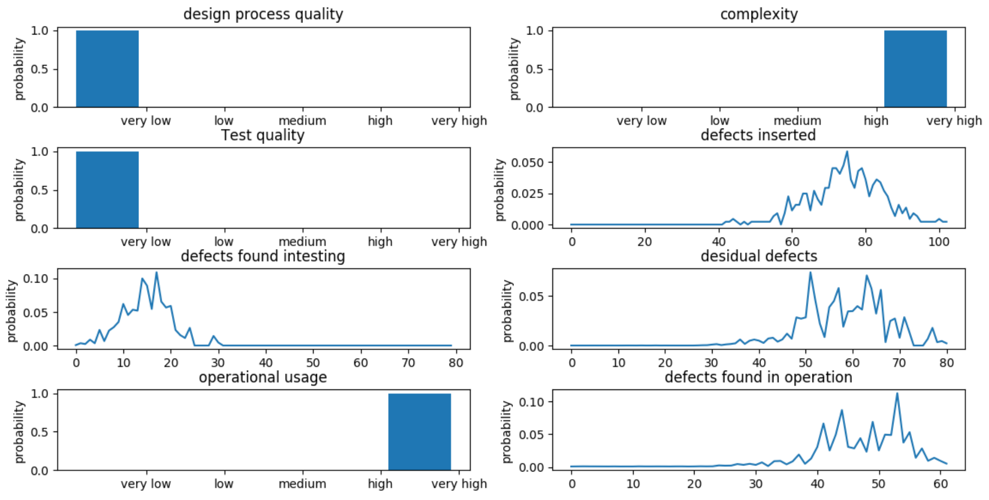


Figure 6.1: Static model - Worst case scenario

The conditional probability function number of defects of this scenario is shown in figure 6.1. In this scenario, the input is 0-4-0-4, which means the Design Process

Quality is very low, the Complexity is very high, the Test Quality is very low and the Operational Usage is very high.

The number of Defects found in operation (DFO) has a distribution with mean around 50 and large variance.

6.1.2 Best case scenario

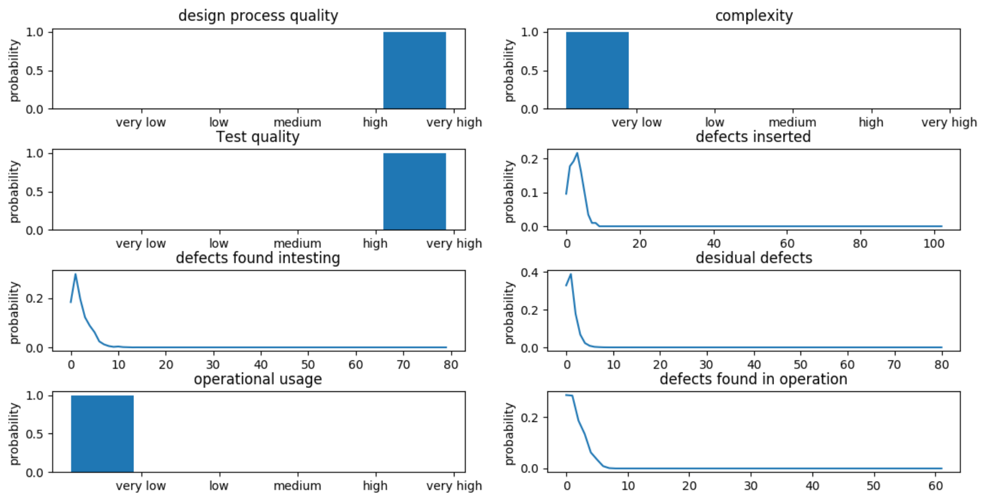


Figure 6.2: Static model - Best case scenario

The conditional probability function number of defects of this scenario is shown in figure 6.2. In this scenario, the input is 4-0-4-0, which means the Design Process Quality is very high, the Complexity is very low, the Test Quality is very high and the Operational Usage is very low.

The number of Defects found in operation (DFO) has a positive half-bell shape distribution with mean around 0 and a small variance.

6.1.3 Average case scenario

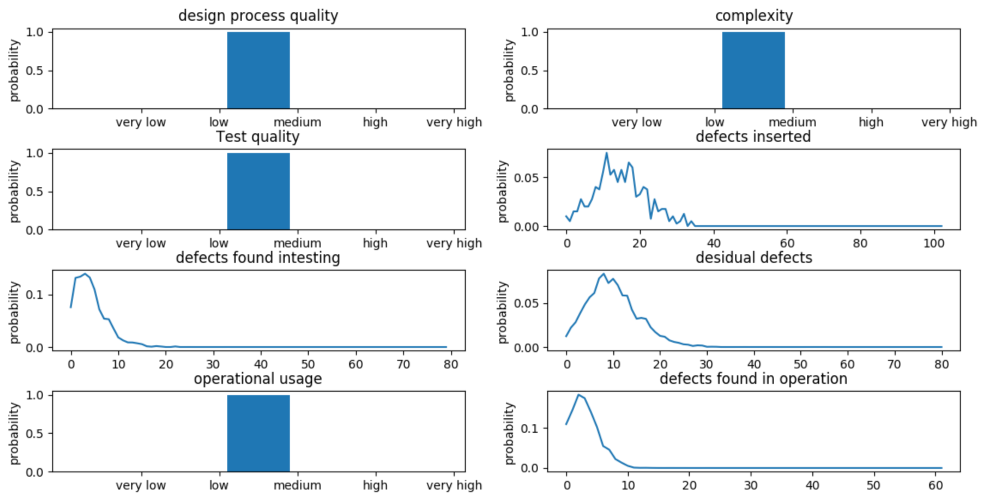


Figure 6.3: Static model - Average case scenario

The conditional probability function number of defects of this scenario is shown in figure 6.3. In this scenario, the input is 2-2-2-2, which means all the parameters are set to medium.

The number of Defects found in operation has a bell-shape distribution with mean around 5 and moderate variance.

6.1.4 No prior knowledge scenario

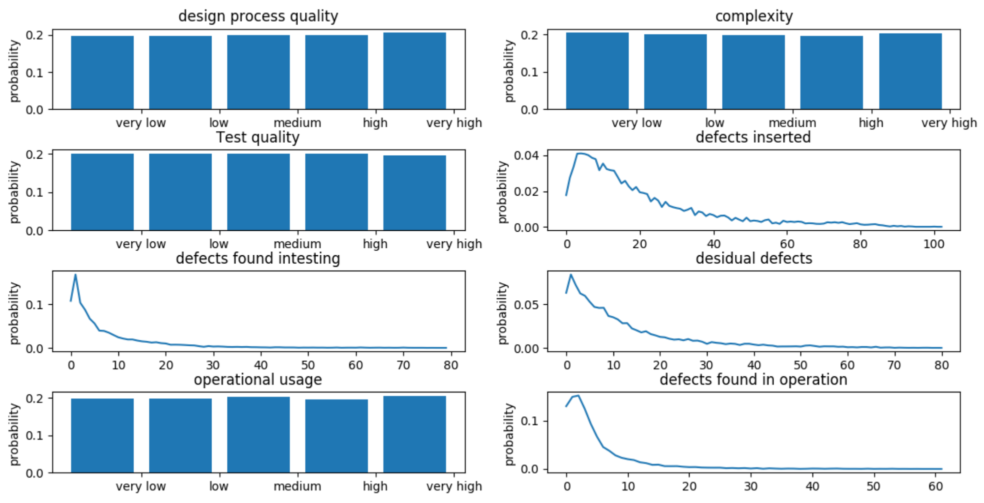


Figure 6.4: Static model - No prior knowledge scenario

The conditional probability function number of defects of this scenario is shown in figure 6.4. In this scenario, all the parameters are unknown, which mean they all have a probability of 0.2 in the range 0-4.

The number of Defects found in operation has a bell-shape distribution with mean around 5 and a very large variance.

6.1.5 Worst Design Process Quality scenario

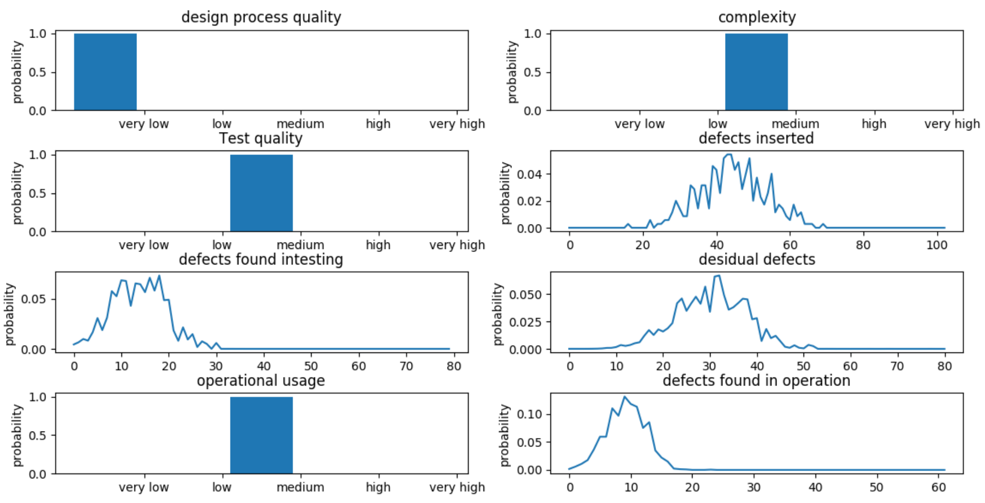


Figure 6.5: Static model - Worst Design Process Quality scenario

The conditional probability function number of defects of this scenario is shown in figure 6.5. In this scenario, the input is 0-2-2-2, which means all the parameters are set to medium except Design Process Quality is set to 0.

The number of Defects found in operation has a bell-shape distribution with mean around 10 and moderate variance.

6.1.6 Highest Problem Complexity scenario

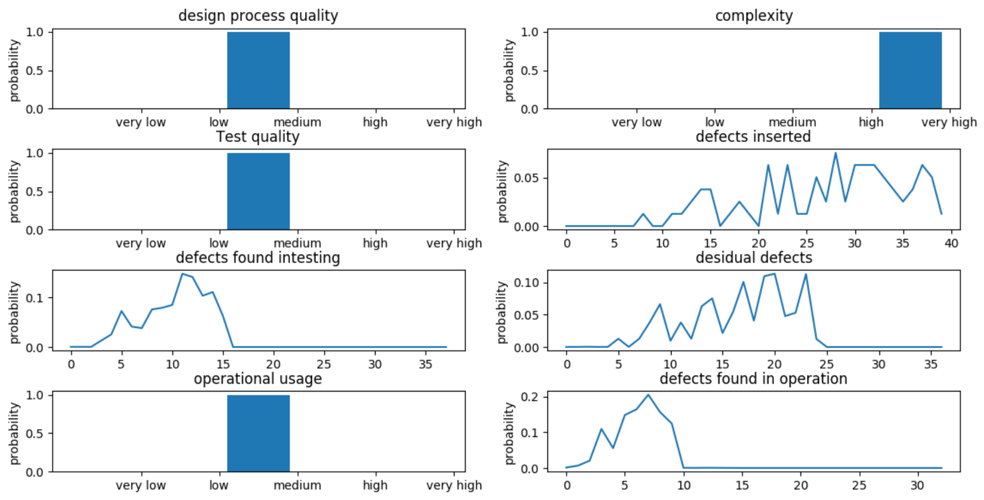


Figure 6.6: Static model - Highest Problem Complexity scenario

The conditional probability function number of defects of this scenario is shown in figure 6.6. In this scenario, the input is 2-4-2-2, which means all the parameters are set to medium except Problem Complexity is set to 4.

The number of Defects found in operation has a bell-shape distribution with mean around 10 and moderate variance.

6.1.7 Lowest Testing Quality scenario

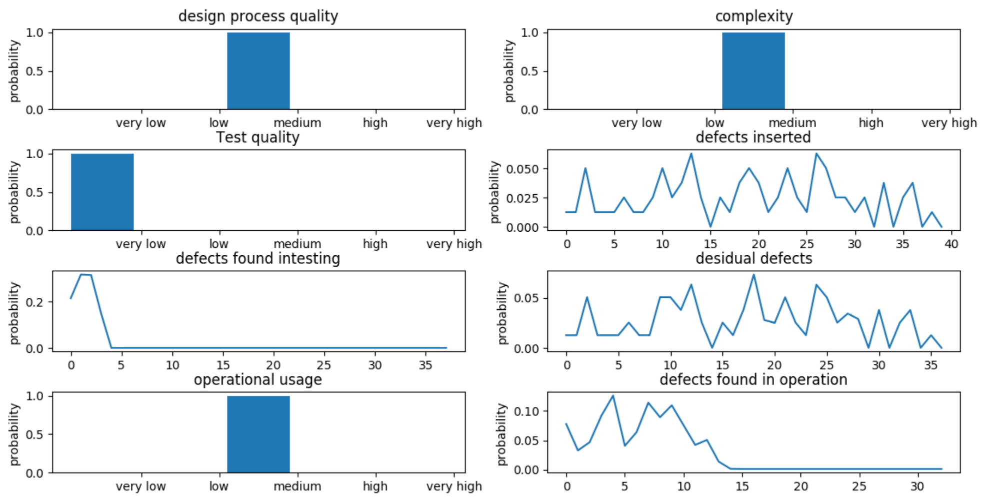


Figure 6.7: Static model - Lowest Testing Quality scenario

The conditional probability function number of defects of this scenario is shown in figure 6.7. In this scenario, the input is 2-2-0-2, which means all the parameters are set to medium except Test Quality is set to 0.

The number of Defects found in operation no longer holds a bell-shape distribution with mean around 8 and large variance.

6.1.8 Highest Operational Usage scenario

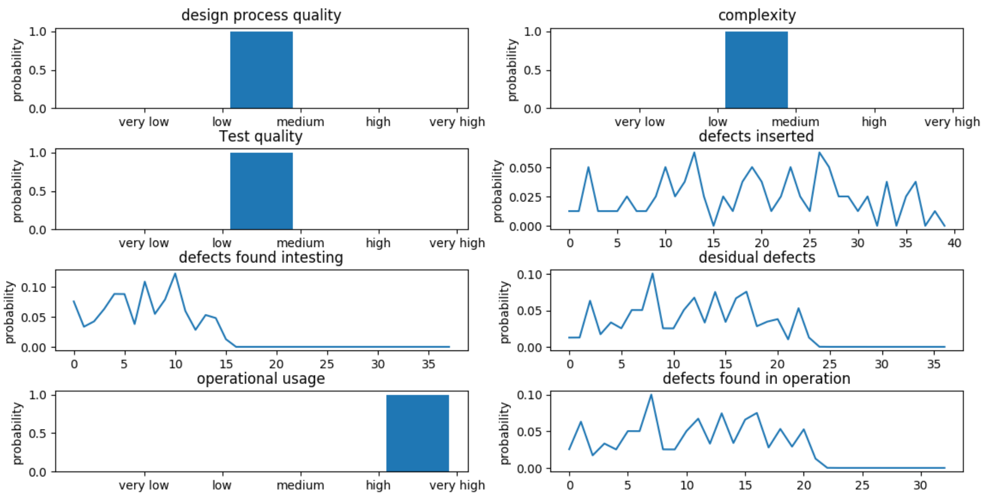


Figure 6.8: Static model - Highest Operational Usage scenario

The conditional probability function number of defects of this scenario is shown in figure 6.8. In this scenario, the input is 2-2-2-4, which means all the parameters are set to medium except Operational Usage is set to 4.

The number of Defects found in operation increases significantly. It no longer holds a bell-shape distribution. It has a mean around 13 and a very large variance.

6.2 Experiments with the Dynamic model

6.2.1 Worst case scenario

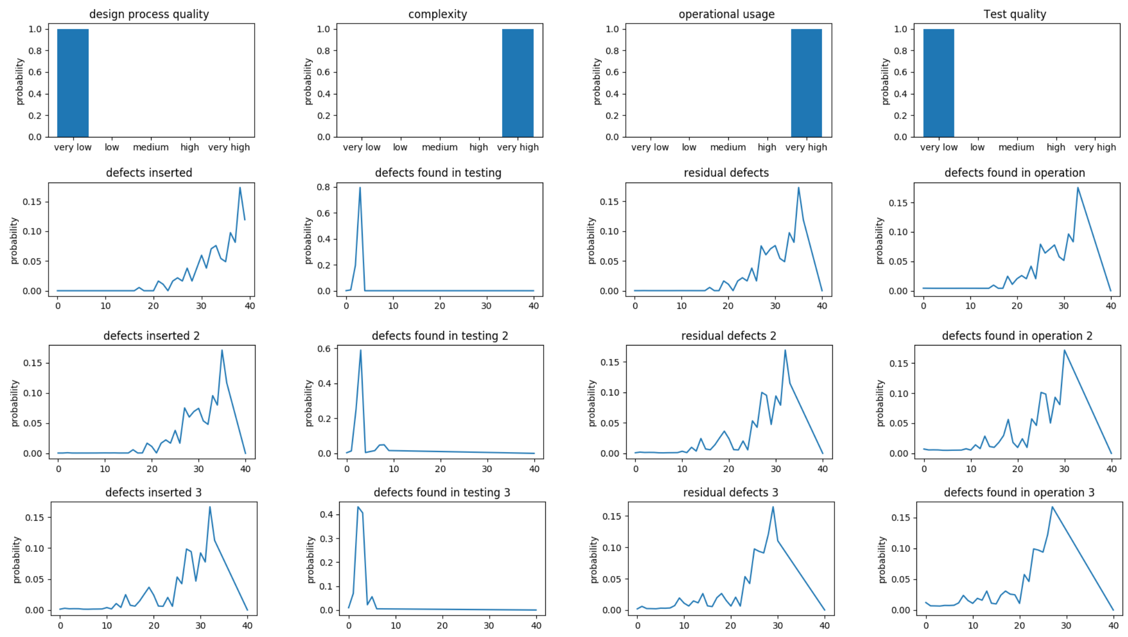


Figure 6.9: Dynamic model - Worst case scenario

The conditional probability function number of defects of this scenario is shown in figure 6.9. In this scenario, the input is 0-4-0-4, which means the Design Process Quality is very low, the Complexity is very high, the Test Quality is very low and the Operational Usage is very high.

Similar to the Static model, the number of Defects found in operation has a distribution with mean around 30 and large variance. Moreover, we can see the numbers of Defects found in operation tends to decrease after each step of software testing.

6.2.2 Best case scenario

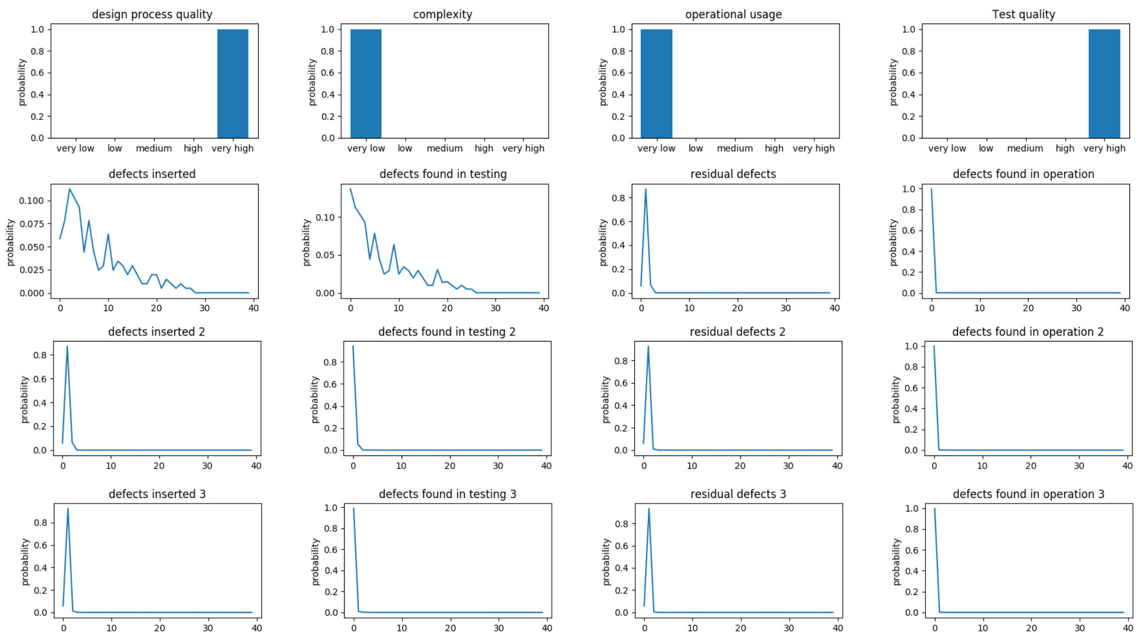


Figure 6.10: Dynamic model - Best case scenario

The conditional probability function number of defects of this scenario is shown in figure 6.10. In this scenario, the input is 4-0-4-0, which means the Design Process Quality is very high, the Complexity is very low, the Test Quality is very high and the Operational Usage is very low.

The number of Defects found in operation has a distribution with mean around 0 and very small variance. The DFO is already very small that after two other testing phases that it does not reduce significantly.

6.2.3 Average case scenario

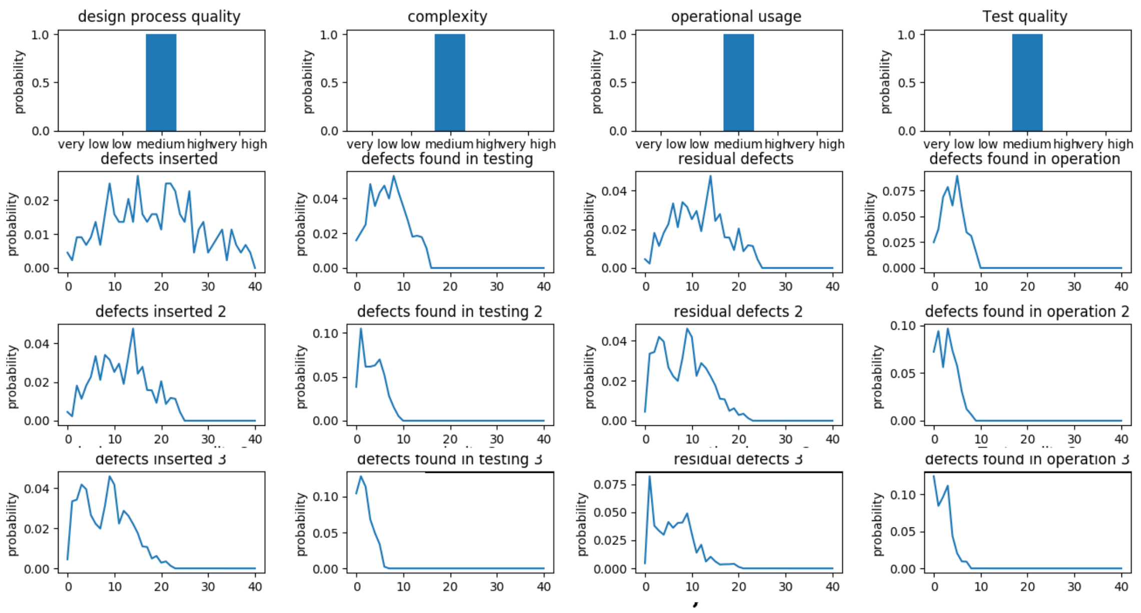


Figure 6.11: Dynamic model - Average case scenario

The conditional probability function number of defects of this scenario is shown in figure 6.11. In this scenario, the input is 2-2-2-2, which means all the parameters are set to medium.

Similar to the Static model, the number of Defects found in operation has a bell-shape distribution with mean around 5 and moderate variance. Moreover, we can see the number of Defects found in operation tends to decrease after each step of software testing.

6.2.4 No prior knowledge scenario

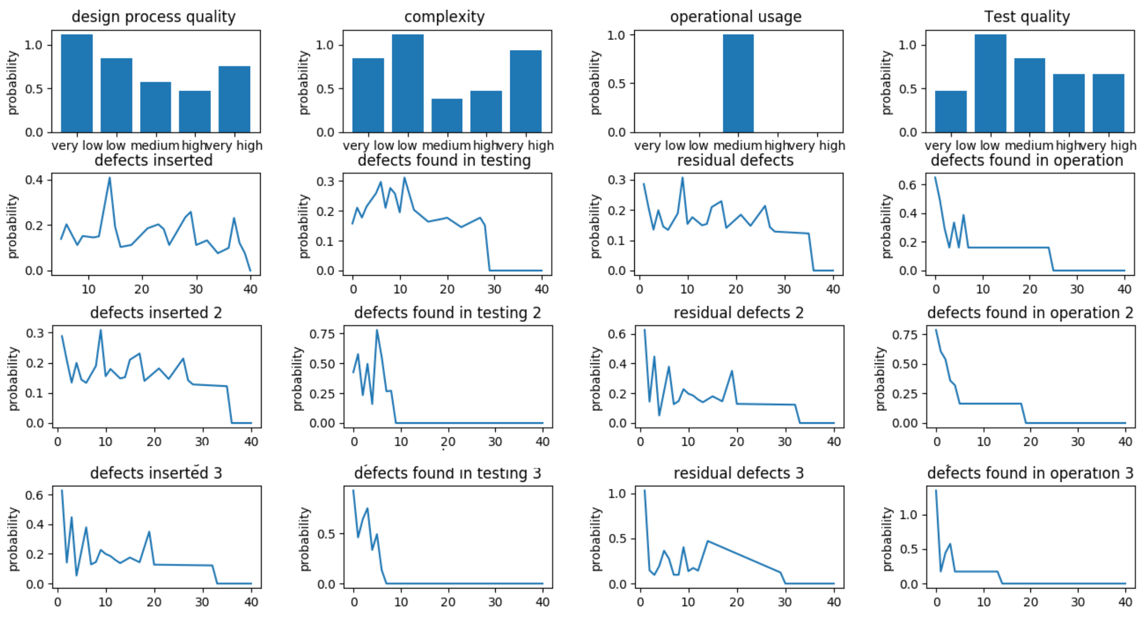


Figure 6.12: Dynamic model - No prior knowledge scenario

The conditional probability function number of defects of this scenario is shown in figure 6.12. In this scenario, all the parameters are unknown.

Similar to the Static model, the number of Defects found in operation has a bell-shape distribution with mean around 5 and a very large variance. Moreover, we can see the number of Defects found in operation tends to decrease after each step of software testing.

6.2.5 Worst Design Process Quality scenario

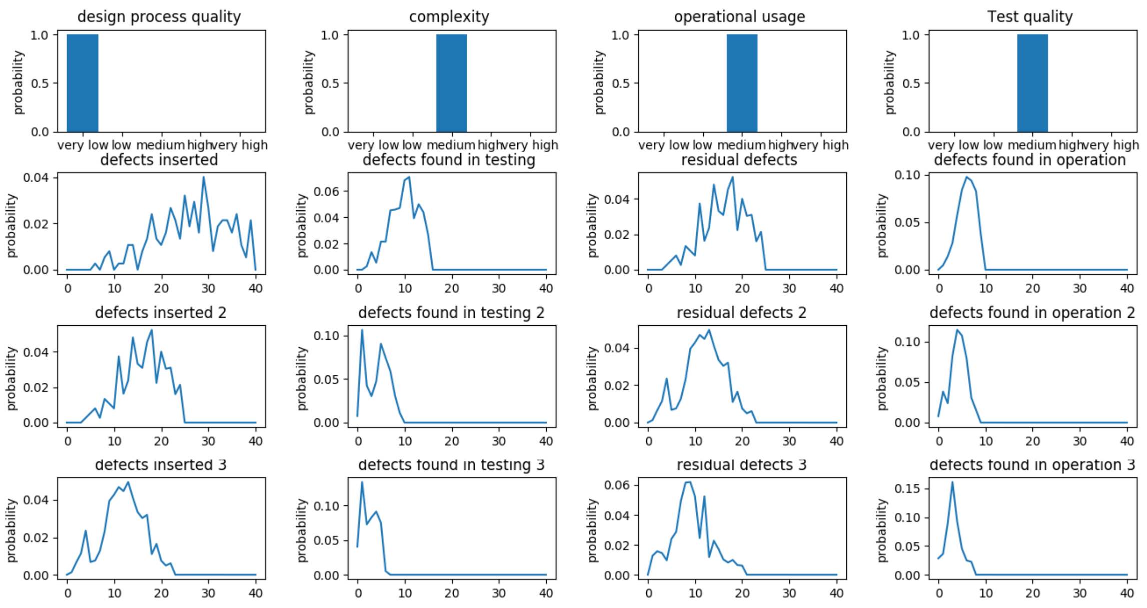


Figure 6.13: Dynamic model - Worst Design Process Quality scenario

The conditional probability function number of defects of this scenario is shown in figure 6.13. In this scenario, the input is 0-2-2-2, which means all the parameters are set to medium except Design Process Quality is set to 0.

Similar to the Static model, the number of Defects found in operation has a bell-shape distribution with mean around 6 and moderate variance. Moreover, we can see the number of Defects found in operation tends to decrease after each step of software testing.

6.2.6 Highest Problem Complexity scenario

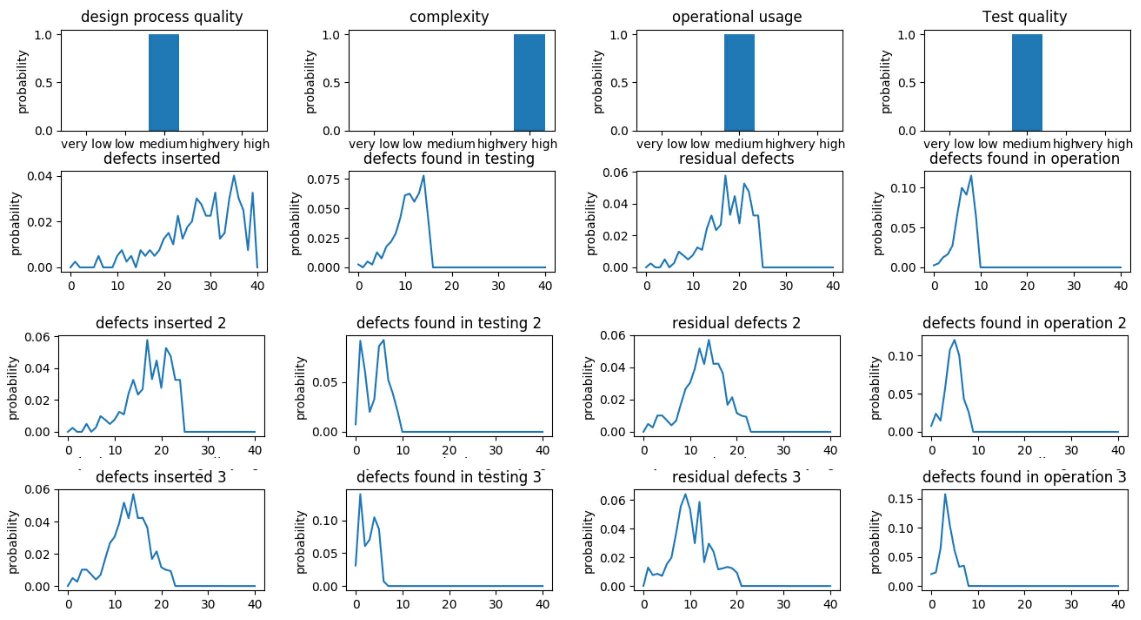


Figure 6.14: Dynamic model - Highest Problem Complexity scenario

The conditional probability function number of defects of this scenario is shown in figure 6.14. In this scenario, the input is 2-4-2-2, which means all the parameters are set to medium except Problem Complexity is set to 4.

Similar to the Static model, the number of Defects found in operation has a bell-shape, the number of Defects found in operation has a bell-shape distribution with mean around 8 and moderate variance. We can also see the number of Defects found in operation tends to decrease after each step of software testing.

6.2.7 Lowest Testing Quality scenario

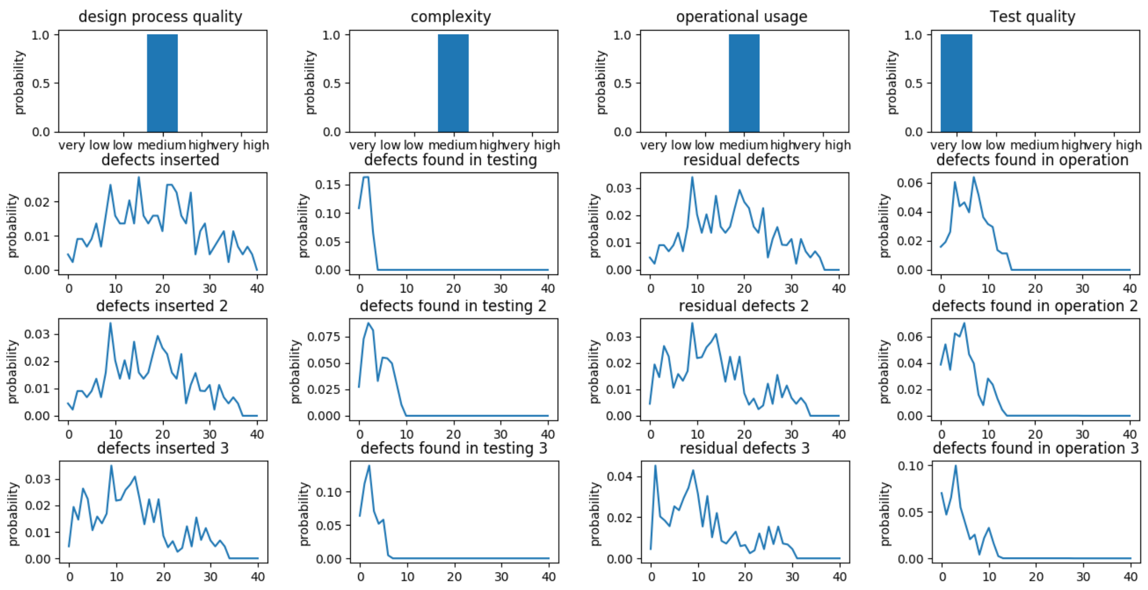


Figure 6.15: Dynamic model - Lowest Testing Quality scenario

The conditional probability function number of defects of this scenario is shown in figure 6.15. In this scenario, the input is 2-2-0-2, which means all the parameters are set to medium except Test Quality is set to 0.

Similar to the Static model, the number of Defects found in operation no longer holds a bell-shape distribution with mean around 8 and large variance. We can also see the number of Defects found in operation tends to decrease after each step of software testing.

6.2.8 Highest Operational Usage scenario

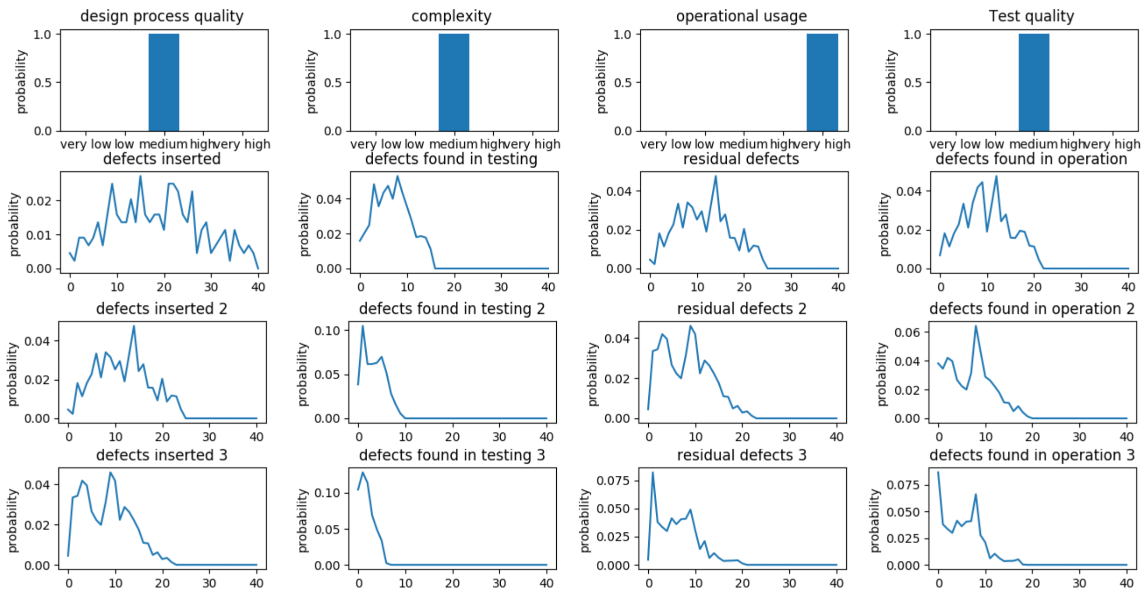


Figure 6.16: Dynamic model - Highest Operational Usage scenario

The conditional probability function number of defects of this scenario is shown in figure 6.16. In this scenario, the input is 2-2-2-4, which means all the parameters are set to medium except Operational Usage is set to 4.

The number of Defects found in operation increases significantly. It no longer holds a bell-shape distribution. It has a mean around 13 and a very large variance. Most interestingly, the effect of testing is so strong that the number of Defects found in operation does not only tend to decrease after each step of software testing but its distribution changes the shape significantly.

6.3 Evaluations

From all experiments above, we can conclude that in overall our proposals for the models incorporates well with the nature of the software development process.

Concretely, in all experimented scenarios, the models give out the correct corresponding qualitative characteristics comparing to the fluctuation of the number of defects logically. In example, the number of defects in the best case should be smaller than in the average and the worst case (**Figure 6.17**) or when the Test Quality decreases, more defects are supposed to be found; when there is one missing input, the variation of the conditional distribution of the number of defects increases accordingly, etc. Most importantly, it also shows the temporal relationship: in all the experiments, the numbers of defects to be found always decrease after each step of testing.

Therefore, it can be concluded that the proposed models are appropriate to be used to estimate the number of defects in a software development process.

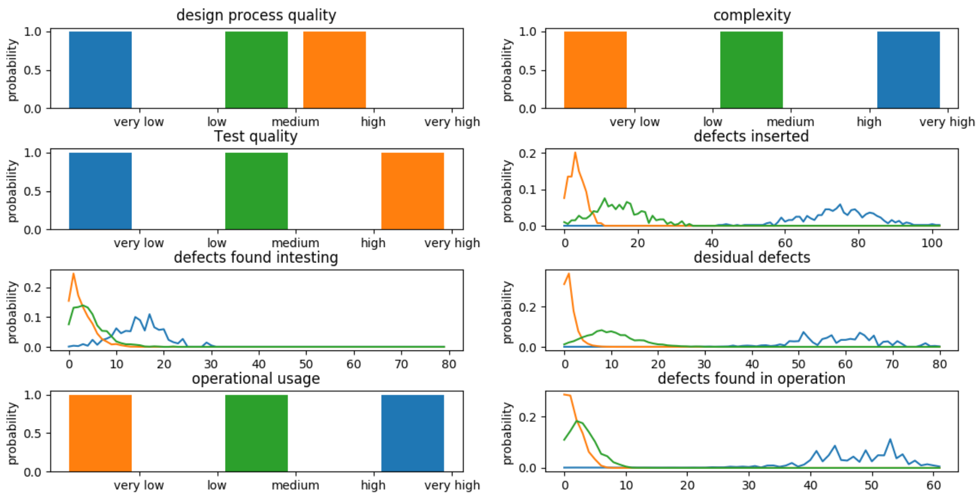


Figure 6.17: (Blue) Worst case (Red) Average case (Green) Best case

Through the experiments, we also exploit the ability of the model to select the most important parameter among the inputs. As it can be inferred from figure 6.18, the most important factor to contribute to this data set is Operational Usage, since when we adjust it to the worst case, it will cause the largest number of defects and also is most unpredictable (its conditional probability has the largest mean and variance).

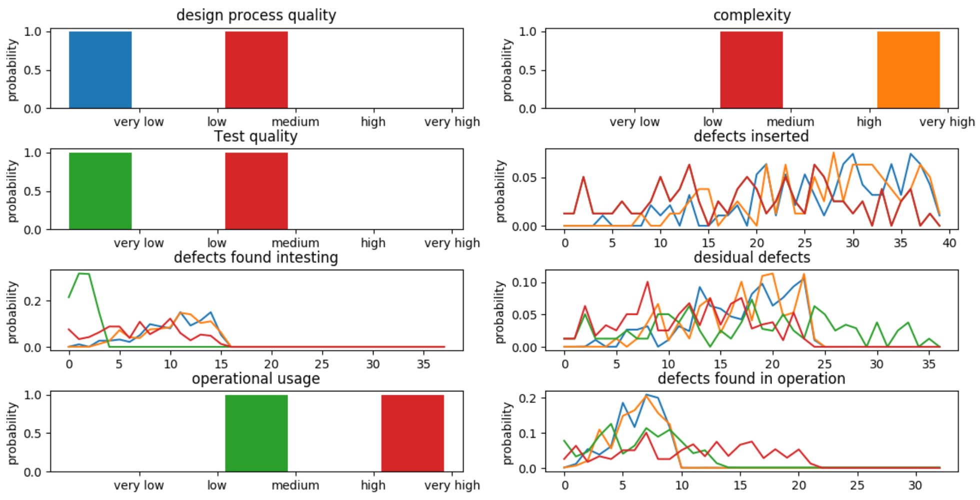


Figure 6.18: (Blue) Worst Design Process Quality (Orange) Highest Problem Complexity (Green) Lowest Test Quality (Red) Highest Operational Usage

Chapter 7

Conclusion

This section concludes the thesis with its conributions, limitations and discuss further development from the approach presented in above chapters.

7.1 Contributions of the thesis

The contribution of this thesis includes several aspects listed as below:

- The explanation in both theoretical and empirical aspects of applying Bayesian Networks for estimating software defects.
- The implementation of an automated system to estimate software defects for better planning.
- The comparison and analyses of different hyperparameters of the proposed model and their effects the model's overall performances.

7.2 Limitation

There are, however, some considerations and possible works that have not been conducted in our work:

- First, since getting the data of software development risks and defects is not feasible for its secrecy, time-consuming and only-available-to-big-software-corporation characteristics, in the scope of this thesis, we have no choice but to use a set of generated data. This led to several drawbacks, especially in learning and confirming the accuracy of the models. This is the primary problem to face in this project.
- Second, we have not proposed a standard metric for discrepancies of each model with different parameters. Having such metric would be useful to give us not only a intuitive but a qualitative measurement to the evaluation process.

7.3 Future work

There are several possible directions from the work in this thesis, if we can have access to the real data, i.e. the data which Fenton collected during his ten-year working in Philips.

The first approach is to build an automatic model using Bayesian structure learning. Instead of proposing a specific model, we can let the computer learn the parameters and the structures of the Bayesian Networks themselves. Then the only task we have to do is picking up the most appropriate model. This, of course will require a large and precise data set which is not always available.

Once we have gotten a good data set, it is also possible to implement other Machine Learning techniques besides Bayesian Networks, such as Neural Networks and compare their performances to see which works the best in this specific problem.

References

[1] Mitsuo Takeda, Hideki Ina, and Seiji Kobayashi, *Fourier-transform method of fringe-pattern analysis for computer-based topography and interferometry*, Journal of the Optical Society of America Vol. 72, Issue 1, pp. 156-160