Cost Benefit Analysis for
Software Process Improvements

Sapan K Shah

Submitted for the Degree of
Master of Philosophy
from the
University of Surrey

UniS

Department of Computing
School of Electronics and Physical Sciences
University of Surrey
Guildford, Surrey GU2 7XH, UK

Oct 2008

Supervised by: Professor Paul Krause, Dr Bill Mitchell

©Sapan K Shah 2008
Abstract

Software Process Improvement (SPI) techniques have repeatedly proven to be effective in removing defects from software artefacts and thereby reducing the costs of the project. Process improvement, however, is not always successful effort. Very few are able to quantify the short and long term costs and benefits of implementing effective SPIs. To solve this, here we propose an approach, which helps not only to implement effective software process, but also to analyse costs and benefits associated with improving the process.

Among many available SPI techniques, this research focuses on Software Inspections. As the initial aim a Bayesian Belief Network model for Inspections effectiveness has been developed. The developed model provides a structure for improving Inspection process in a disciplined and consistent way. We can say that the model can improve the likelihood with which a software organisation can achieve its cost, quality, and productivity goals. At the end, a probabilistic methodology for cost benefit analysis is proposed. By utilising such methodology, research can conclude which variables are keys to implement cost effective process and can also provide the idea for the decision-maker if the change in the process is valuable of carrying out.
Acknowledgement

The best way to have a good idea is to have a lot of ideas. I sincerely feel that the credit of the research work could not be narrowed down to only one individual. The production of this thesis involves many valuable contributions. This work is an integrated effort of all those concerned with it, through whose co-operation and effective guidance I could achieve its successful completion. My friend Mr Kaumil Shah says a big writing project is like having a hungry and bad-tempered monster chained up in your basement: you can go out and have fun, but eventually you must return home and feed the hungry beast. There are many people who have helped me feed this beast over the past few years who deserve my sincere thanks for their patience, their contributions, or both. This thesis is dedicated to all those people who have unwittingly provided me with both the material and the motive to write it.

I am thankful to our Head of department Professor Steve Schneider, for providing me necessary support. I thank to University of Surrey, for providing me all facilities and other technical materials and a good platform for a bright career. I am also thankful to Prof Tony Ho and Dr Matthew Casey for being a constant source of inspiration for me and sharing their ideas with me. I also extend my gratitude towards all other faculty members of computing department for their help and support.

I would like to express my heartily gratitude to, Professor Paul Krause and Dr Bill Mitchell, my Project Supervisors for giving me encouragement and support for my research work. To them I owe more than what I can mention… mostly for teaching me to see the silver lining in every dark cloud. They both kept faith in me and considered me for challenging research-work to prove myself and endeavoured me from learning & possessing the knowledge during the whole period. During the entire period, I have received endless help from them, right from deciding the research scope, collecting the
requirements, designing thesis format, and what not.… In many cases, they were the first persons to vet my terms and theories. They supported me and reassured me and read all of the thesis-drafts throughout the monster-feeding process. They politely pointed out successes and failures in the research documents. Even during the last days, they helped me and gave their valuable time to provide a moral support, without which this research would be, a dream.

Several individuals from industry also made enormous contributions to the project. Without a doubt, I would like to thank Mr Derek Treagus, Mr Capers Jones, Dr Elena Perez-Minana, Dr Rishabh Gupta, Mr Nil Malotaux, Mr Aakash Saxena, Dr Don O’Neil, Mr Ram Chillarege, Mr Gerold Keefer, Mr Patrick O’Toole and Mr Otto Vinter for cooperating with me and also giving me some data to clear idea of the research requirements.

I am also grateful to all those online friends at qaforum.com and spi.yahoogroups.com whom I never met, but over the Internet who have shared their enthusiasm and knowledge for process improvement with me, those who have challenged me to search for greater relevance, and particularly those whose criticism have made me see beyond the looking-glass.

Thanks to friends Vinod Jadav, Harish Jeswani, Manas Swain, Arpit Patel, and Gaurang Mistry also for being the helping arm during the project work, providing me invaluable materials and sharing their knowledge with me. I also thank Dr Hardik Shah, and Miss Nehal Solanki, for giving me guidelines to prepare a good thesis. I am grateful to them for providing critical feedback and support where deserved.

There are times in such projects when the clock beats you time and again and you run out of energy and you just want to finish it once and forever. My Parents and my brother made me endure such times with their unfailing humor and warm wishes. I heartily thank them who taught me to look. Their encouragements have made me what I stand today. Above all, I thank God for the gift of life, friends and family.
List of Figures

Figure 2.1  Software Inspection Process  21  
Figure 3.1  Graphical Network  32  
Figure 3.2  The Causal Network for Wet Grass Situation  34  
Figure 3.3  Serial Connection Pattern  35  
Figure 3.4  Diverging Connection Pattern  36  
Figure 3.5  Converging Inspection Pattern  37  
Figure 3.6  Salmonella and flu may cause nausea, which in turn causes pallor  39  
Figure 3.7  A Causal Network  43  
Figure 3.8  A Causal Network  44  
Figure 3.9  A Bayesian Network Representing Causal Influence  46  
Figure 3.10  Nodes Represent Random Variables in BBNs  47  
Figure 3.11  Edges Represent the Causal Relations in BBNs  47  
Figure 3.12  A BBN Consisting of Graphical Networks and Conditional Probability Table  48  
Figure 3.13  Basic Nodes of Influence Diagram  51  
Figure 3.14  Connections of Influence Diagram  52  
Figure 3.15  Influence Diagram Example – Initial State  53  
Figure 3.16  Influence Diagram Example – with evidence  54  
Figure 3.17  Using Influence Diagram for Process Improvement  55  
Figure 3.18  Using Influence Diagram for Process Improvement – with NPT  56  
Figure 4.1  BBN for Software Inspection Effectiveness – Model 1  61  
Figure 4.2  BBN for Software Inspection Effectiveness – Model 2  62  
Figure 4.3  BBN for Software Inspection Effectiveness – preferred model  63  
Figure 4.4  Nodes affecting Difficulty of Work Product  69  
Figure 4.5  NPT for Quality of Logging  78
Chapter 1

Introduction and Research Objectives

1.1 Background

*The role of software becomes increasingly critical for business as well as for human lives.*

(Jones, 1996:23)

The problems caused by software products that are late or over budget, or that do not work, are very common. Developing reliable software on time and within budget represents a difficult endeavour for many organisations. Loss of life or widespread inconvenience caused by unreliable software makes big headlines in the news media. For example, failure of Ariane 5 Flight 501, which was mainly because the working code for the Ariane 4 rocket was reused in the Ariane 5 without proper analysis (Wikipedia, 2008). Improved software quality is essential to avoid these failures, to ensure reliable products and services, and to gain customer satisfaction.

Structured methods were developed in the 1970s to achieve software quality. That was the first wave of the software industry. It came as a response to the growing need to build complex interactive commercial applications using shared systems and make such systems maintainable. According to Zahran (1998), structured methods focus on ways to formalise the definition of requirements and on the traceability of requirements from the design to the finished system stage. Although this was the beginning of transforming
software development to mass production, it was not quite enough. Real issues that make or break software projects, such as project management and requirements management, were not a mainstream focus.

The software process movement came as a response to the increasing rate of failure of software projects. Focus on process started through sponsorship by the US Department of Defense (DoD) which funded the Software Engineering Institute (SEI) to come up with a method for assessing the capability of the Department’s software subcontractors (Fuggetta & Wolf, 1996; Humphrey, 1989). Since the 1980s the process message coming out of the SEI has gone from strength to strength to influence the whole software industry worldwide. As the result, we are now in the midst of the Software Process Improvement (SPI) wave. There are number of international standards and initiatives for the SPI such as the CMM (Capability Maturity Model), ISO (International Standards Organisation model), and so on. The SPI literature is full of evidence from successful companies and descriptions of their SPI programs, e.g., Hewlett-Packard (Grady, 1997), Motorola (Daskalantonakis, 1992), NASA (Basili et al, 2002), Philips (Roijmans, 1996), Raytheon (Dion, 1993), Siemens (Mehner, 1999), and so on.

SPIs offer potential benefits, but also cost a lot. Many companies have invested large sums of money in improving their software processes, and several research papers document SPI’s effectiveness (Basili et al, 2002; Conradi & Fuggetta, 2002; Curtis & Statz, 1996; Dyba, 2005; Goldenson & Herbsleb, 1995; Humphrey et al, 1991; Karistrom et al, 2002; Stelzer & Mellis, 1998; Solingen, 2004; and many more). It is clear that without quantifying costs and benefits, it would be impossible to properly decide whether SPI is worth its cost. Analysing SPI’s costs and benefits is relevant for: estimating how much effort to invest to solve a certain problem or estimating whether a certain intended benefit is worth its cost. The purpose is also to obtain the Return On Investment (ROI) number for communication and decision purposes (Rico, 2004).

For example, the National Software Quality experiment (O’Neil, 2001) has collected practitioner data on SPI since 1992. Out of the 78 organisations that participated, most
achieved a ROI between 2:1 and 8:1. This clearly shows that impressive results are possible, but results also say that still there is lack of consistent ROI. This leaves a negative impression in the minds of the participants, and also shows that badly conducted SPI programs do not pay for themselves. To gain most return of investment, it is important to implement the most effective process possible for particular program.

Variations on the techniques have been proposed to do the cost benefit analysis. Rico (2004) and O'Neil (2005) explain use of formulas like Benefit/Cost ratio (B/CR) and Net Present Value (NPV) to analyse costs and benefits associated with different SPI models (chapter 2). However, these techniques seem inadequate for various reasons – like they only describe statistical relationships; fail to express depth of detailed process or to consider uncertainty associated with the software process. We also have a few process models, which would ultimately improve process quality. However, the question is that how much does quality cost? Quality provides benefits, but also incurs costs, and the two are interrelated and inseparable; and very few are able to achieve cost effective quality. To solve this, here we propose a methodology, which would not only help implement high quality software process, but also to analyse costs and benefits associated with the improvement efforts.

1.2 Proposed work: Aim and Objectives

Conventional approaches for cost benefit analysis fail to provide real quantitative support for decision-making. In comparison to these, here we aim to develop a graphical executable model by considering relations between interrelated process influence factors; which can be used to analyse return on investment for SPIs.

We use a specific Artificial Intelligence (AI) technique called Bayesian Belief Network (BBN) to model and manage software process uncertainties. Detailed reasons for this choice are provided in chapter 3. The systematic approach proposed in this research ensures that human factors are included in the model rather than factored out. The
proposed model can help identify high risk zones in the software process, and thus guide and ultimately improve decisions based on changes in the process.

As explained above to gain most return of investment, it is important to implement the most effective processes. Therefore as an initial hypothesis, it has been explained here that the model can be used to implement effective process (chapter 4), which also provides a platform for cost effective improvements (chapter 5).

The main research question of this thesis is: using the application of Bayesian Belief Networks, how can we analyse the costs and benefits of implementing effective SPI techniques?

1.3 Hypothesis

The principal hypotheses investigated in this research are:

1. Bayesian Belief Networks can be used for estimating the effectiveness of SPI activities, particularly Software Inspections.

2. By applying the proposed cost benefit analysis phase model, available resources can be better utilised, thus making the software development process more cost-effective.

Initially, only Inspections were analysed, and few of the questions that can be answered by using proposed Bayesian Belief Network are (chapter 4): How much calendar time should Inspections take? What is the quality of the inspected software? How effective are Inspections? How much do Inspections cost? What is the contribution of the experience of the moderator and other Inspection team members to the effectiveness of the Software Inspection process? After that research restructured all_activities network developed by Agena (Fenton & Neil 2004), which successfully uses formally defined, collected and classified defect data; can be applied at any phase of software development; and is not only able to predict defect rates at various phases, but also helps
to highlight the decisions or optimise available resources, where they are most cost effective (chapter 5).

1.4 Potential Benefits of Proposed Work

The application of the developed model provides industry with a new and more efficient way of improving software processes, and in concentrating on the more important attributes of the process to gain more benefits from invested efforts.

The proposed model for cost benefit analysis contributes to achieve software productivity and quality. The methodology recognises the uncertainty associated with process improvement techniques. Advantages of this approach are: it can consider the interactions among variables, and can highlight key variables and their possible implications; once these variables are identified, it may be possible to modify the process to gain maximum benefits.

Researchers and practitioners can profit from this research in different ways. Most importantly, the application of Bayesian Belief Networks can offer visibility into the ways in which resources relate to one another. It can help to determine the amount of money to be gained or lost by creating and using a new software process. Thus, it can provide support for decision-making: whether to use a new process, or revert to an old one, or to make more modifications to achieve desired benefits or improvements.

1.5 Research Philosophy, Approach and Strategy

The research philosophy is interpretivist as research findings arrive from both experimental laboratory and real-world settings (Remenyi, 2000). The research phenomenon in this investigation utilises available literature for process improvement; such as Software Inspections, defect classification schemes; how better process improvement decisions are made to gain maximum return on investment. The approach
comes closest to deductive reasoning (Saunders, 2003; Remenyi, 2000), where one starts thinking about generalisations, and then proceeds towards the specifics of how to prove or implement the generalisations. The approach is applicable to this research as in the disciplines of process improvement agreed facts and established theories are available. Here using collected quantitative data and scientific principles, research wants to explain causal relationships between process improvement variables. The aim is to propose highly structured approach by selecting samples of sufficient size in order to generalise the methodology. Here, research wants to answer why reliable support for process improvement is necessary and how our proposed methodology can help answer the research question. Therefore, the research strategy is Experimental (Yin, 1994), which also focuses on contemporary software engineering principles. Research designed the model, and then executed the model and analysed executed output. Regarding the validation of the developed model, experiments are considered. However, detailed experiments could not be possible for two main reasons. In the first place, it was not possible to validate complete methodology using academic laboratory settings. And secondly, it was not possible to find enough organisations and individuals who could collaborate in such independent experiments. Data collection methods used are questionnaires and document analysis/study.

<table>
<thead>
<tr>
<th>Layer</th>
<th>Exploration Phase</th>
<th>Design Phase</th>
<th>Validation Phase</th>
</tr>
</thead>
<tbody>
<tr>
<td>Research philosophy</td>
<td>Interpretivism</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Research approach</td>
<td>Deductive</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Research strategy</td>
<td>Experimental</td>
<td></td>
<td>Experimental</td>
</tr>
<tr>
<td>Data collection method</td>
<td>Questionnaires,</td>
<td></td>
<td>Questionnaires,</td>
</tr>
<tr>
<td></td>
<td>Documentations</td>
<td></td>
<td>Documentations</td>
</tr>
</tbody>
</table>

Table 1.1: Adopted research philosophy approach and strategy in this research
1.6 Thesis Outline

The main topics covered in each of the chapters are as follows:

- **Chapter 2** mainly deals with the **Literature Review** and lays the intellectual foundation for the rest of the research. Firstly the chapter defines and explains basic concepts of **Software Process Improvements**. Then in the chapter, available literature on **Software Inspections** has been summarised, as research proposes a model to improve effectiveness of Software Inspections.

- Then, **Chapter 3** outlines the essential features of **Bayesian Belief Networks**, as research proposes the use of BBNs to solve the problem. At the end of the chapter, an example using an **Influence Diagram** is discussed, which gives the case for the use of the Bayesian technique to analyse costs and benefits of SPIs.

- Then **Chapter 4** gives detailed explanation of the development of a **Bayesian Belief Network for measuring the Software Inspections Effectiveness**, which mainly considers how different factors affecting the Inspection process are related. The model described adequately answers the basic hypothesis set out previously, and also lays the foundation for work explained in **Chapter 5**.

- In **Chapter 5**, the model developed using probabilistic notions of Bayesian Belief Networks is proposed, which can provide support for implementing effective software process in a more mature way and can also be used to analyse return on investment for process improvement. By offering structure and using classified defect data research facilitates the early identification of problems that threat decision success and thus give better cost effective solution. The emphasis of the structured model is on reducing the risks associated with decisions and optimising available resources by assigning them where they give maximum benefits.

- Finally, **Chapter 6** gives **Conclusions**, summing up the complete thesis.
Chapter 2

Software Process Improvements

Abstract

Use of procedures or standards is necessary to achieve satisfactory software quality. However, in order to judge whether practices are cost effective at achieving the required reliability of the product, a measurement-based improvement approach to software development is required.

Relying on single process improvement activity to achieve the required level of quality is also not a cost efficient process. To improve or implement the cost effective process, a number of techniques have been proposed. Improvements, using tools, have been suggested to help with the cost benefit analysis of software processes, and more particularly for specific phase of the project; although the current research is incapable to help for making reliable cost-effective decisions.
2.1 Introduction

Humphrey (1989) explains that software process is a set of activities, methods, and practices used in production and evolution of software. The process defines the way we act or react in certain situation, or the activities to fulfil a certain task. The common process thinking across a group of individuals aligns the behaviour and activities of those individuals towards achieving their common goal. It brings consistency and uniformity to the group's behaviour, which turn into improved capability and better quality of results.

There are a number of organisational roles and mechanisms that should be in place for the process to be effective (Zahran, 1998). To start with, the process must be defined. Then the knowledge of the process must be passed to those who will perform it. Then the defined process should be followed and measured consistently to achieve the improvement. There should also be some feedback loops to achieve a continuously improving process environment. Analysis of such feedback helps the development team and their management trace the reasons for poor quality and identify areas for improving the process and product quality.

Software industry has been following this ‘process’ oriented improvement approach from last decade and literature is full of evidence from successful companies and descriptions of their process improvement programs, e.g., Hewlett-Packard (Grady, 1997), Motorola (Daskalantonakis, 1992), NASA (Basili et al, 2002), Philips (Rojmans, 1996), Raytheon (Dion, 1993), Siemens (Mehner, 1999), and so on.

These case studies prove that software process improvement is necessary to achieve not only satisfactory software quality but also to reduce release time and cost of the development as well. However, sometimes these practices themselves cost a fortune, and in order to judge whether practices are effective at achieving the required return on investment, a measurement-based approach is required. Here research proposed such approach which can be used for estimating the effectiveness of the process
improvement, as well as to concentrate on the more important attributes of the process to gain more benefits from invested efforts.

2.2 Software Process Improvements and Measurement

Software development is a discipline with specific management difficulties. Collecting relevant data during development is one way to overcome these difficulties. Such data collection for software development is termed as Software Measurement (Berka, 1983:9).

Measurement is essential for understanding, defining, managing, controlling and improving the software development and maintenance processes. According to Berka, to have a deep understanding of software development activities and their interrelationships, one must characterise the various aspects of improvements in a quantitative way. In turn, one can use the measurements that result to set goals for productivity and quality and to establish a baseline against which improvements are compared. During development, measurements can point to hot spots that need further attention, analysis or testing. Even during maintenance, measurements could reflect the effects of changes in size, complexity and maintainability. The measurements also support planning, as projections and predictions about future projects can be made based on data collected from past projects. Measurements can be used to assist in evaluating tools and strategies and in tailoring the development process and environment to the situation at hand. Most importantly, measurements offer visibility into the ways in which processes, resources, methods and technologies relate to one another.

Measurement plays a critical role in project management, process understanding, and process and product improvement. With measurement, developers can evaluate current situations and products, predict future characteristics and behaviour, and control the development and maintenance processes. Derived from what is visible in the process, the measurements are related to the maturity of the development process. At the same
time, the measurements help to increase visibility and understanding of the process, thereby leading to process improvement.

However, both management and software engineers often spend some additional time and labour needed to support data collection and analysis. For example, the Software Engineering Laboratory at NASA’s Goddard Space Flight Centre reports that data collection and analysis add 7 to 8% to the cost of a project, and DeMarco (1982) estimates that development costs increase between 5 and 10% when measurement data collection is involved. As explained previously, measurements improve quality – but, is it really worth spending big bucks and more importantly large effort & time to gain this quality? More than that, all different measurement approaches cost differently and offer different degree of advantages. How exactly can one know from where to start and what will be gained after couple of years? We believe that, only then, when measurements are clearly needed and relatively easy to understand, will measurement be a welcome part of software development and improvement processes; and therefore it is necessary to propose a methodology that can be used for this analysis.

2.3 Software Metrics

As mentioned above, effective management of any process requires quantification, measurement, and modelling. Software metrics provide a quantitative basis for the development and validation of models of the software development process. Metrics can be used to improve software productivity and quality. Software metrics and models have been proposed and used for some time and the results (Wolverton, 74; Perlis, 81) indicate that the careful implementation and application of a software metrics program can help achieve better management results, both in the short run (for a given project) and in the long run (improving productivity on future projects).
Essentially, software metrics deal with the measurement of the software product and the process by which it is developed. Software metrics can be classified into three categories: product metrics, process metrics and project metrics. Product metrics are those that describe the characteristics of the product such as size, complexity, design features, performance, and quality level. Process metrics are those that can be used for improving the software development and maintenance process. Examples include the effectiveness of defect removal during development, the pattern of testing defect arrival, and the response time of the fix process. Project metrics are those that describe the project characteristics and execution. Examples include the number of software developers, the staffing pattern over the life cycle of the software, cost, schedule, and productivity.

2.3.1 Software Quality Metrics

To develop the cost benefit analysis model for software process improvements—research focuses on Software Quality Metrics, which are mainly subset of software metrics that focus on the quality aspects of the product and process. Nonetheless, the project parameters such as number of developers and their skill levels, the schedule, the size, and the organisation structure certainly also affect the quality of product.

Software quality is a characteristic that theoretically at least can be measured at every phase of the software development cycle. One can generate long list of quality characteristics for software—correctness, efficiency, portability, maintainability, reliability, etc. Early examples of work on quality metrics are discussed by Boehm and McCall (McCall; 77). However, it has been noticed that the characteristics are often overlapping and conflict with one another; for example, increased portability may result in lowered efficiency. Therefore, useful definitions of general quality metrics are difficult to create, and almost all researchers have neglected efforts to find any single metric for overall software quality.
Although a great amount of research is done in this area, it demonstrates less commonality of definition than other areas of metric research. Three areas that have received considerable attention are: Program correctness (as measured by defect counts); software maintainability (as measured by various metrics like complexity); and software reliability (as measured from defect data) (Cerino, 1986). Examples of these areas are briefly discussed below:

- **Defect Metrics:** The number of defects in the software product should be readily derivable from the product itself; thus, it qualifies as a product metric. The number of defects observed in a software product provides, in itself, a metric of software quality. Studies have attempted to establish relationships between this and other metrics that might be available in the development cycle, and that might, therefore, be useful as predictors of software quality (Conte, 1986; DeMarco, 1982). Some examples of defect metrics are number of design changes defects, number of defects per LOC, ratio of defects during system testing, and so on.

- **Maintainability Metrics:** A number of efforts have been made to define and implement metrics that can measure or predict the maintainability of the software product (Yau, 1980; Yau, 1985). For example, an early study by Curtis and his group, investigated the ability of Halstead’s effort metrics to predict the psychological complexity of software maintenance tasks (Curtis et al, 1979). Assuming such predictions could be made accurately, complexity metrics could then be profitably used to reduce the cost of software maintenance (Harrison, 1982). Rombach (1987) has also published the results of a carefully designed experiment that indicates that software complexity metrics can be used effectively to explain or predict the maintainability of software in a distributed computer system. The complexity metrics studied included both measures of the internal complexity of software modules and measures of complexity of interrelationships between software modules. The study indicates that such metrics can be quite useful in measuring maintainability and in directing design or redesign activities to improve software maintainability.
• Reliability Metrics: It would be useful to know the probability of software failure, or the rate at which software errors will occur. Again, although this information is inherent in the software product, it can only be estimated from data collected on software defects as a function of time. If certain assumptions are made, these data can then be used to model and derive software reliability metrics (Jones, 1993). These metrics attempt to measure and predict the probability of failure during a particular time interval, or the mean time to failure (MTTF). Since these metrics are usually discussed in the context of developing a reliability model of the software, more discussion of these metrics is given in the chapter 3 (Bayesian Belief Network for Software Inspections Effectiveness).

2.4 Software Reliability Models

Independent of the fact how the process improvement process is organised (Beizer, 1995; Hetzel, 1993), at some point in time the question arises how reliable the software product is – means, how long will the software run before it fails and how expensive will the software be to maintain? Reliability, defined as the probability that a product would operate without failure under given conditions for a given time interval, is an important non-functional requirement to take into account when the product quality question is raised. As noted before, if testing of the last project stage is stopped too early, significant defects would be released to its intended users and the software manufacturer would incur the post-release cost of fixing resulting failures afterwards. If testing proceeds too long, the cost of testing and the opportunity cost could be substantial. Littlewood and Strigini point out that not focusing on software reliability prediction and estimation can cause serious problems - both in achieving sufficient reliability and also demonstrating software's achievement (Littlewood and Strigini, 2000).
From the literature, two types of software reliability models can be distinguished (Reliability Analysis Centre, 1996):

- **Software reliability prediction models** address the reliability of the software early in the life-cycle at the requirements, design or coding level, using historical data. The reliability is for instance predicted using fault density models and uses code characteristics such as lines of code, nesting of loops, etc. to estimate the number of faults in the software.

- **Software reliability estimation models** evaluate current and future reliability from faults gathered beginning with the integration or system testing of the software. The estimation is based on test data. These models attempt to statistically correlate defect detection data with known functions such as an exponential function.

In this thesis, no attention is paid to estimation models, but only prediction models – as the aim is to propose a methodology for cost benefit analysis of SPI, rather than cost estimation. Software reliability prediction models use characteristics of the software and the software development process throughout the development cycle and extrapolate to operational behaviour. Some examples are:

- **Phase-Based Model (Gaffney and Davis, 1988):** This model assumes that faults in the different development phases follow a Rayleigh density function (Probability Distribution Function). Further assumptions are that the staffing level is directly related to the number of faults discovered during development and that estimates for the code size are available during the early phases of the development cycle.

- **COQUALMO (Chulani, 1999):** This model consists of two sub-models, namely the Defect Introduction (DI) and Defect Removal (DR) models. The DI-model is formulated using the product, process, computer and personnel attributes (based on Boehm’s Cost & Schedule estimation model - COCOMO II) and predicts the number of faults to be introduced in the different development phases. The DR-
model estimates the number of faults removed by several defect removal activities (like reviews).

- **Orthogonal Defect Classification (Chillarege, 1992):** This concept enables in-process feedback to developers by extracting elements on the development process from software defects. The methodology classifies software defects and provides a set of concepts that supports guidance in the analysis of defects data. 'Orthogonal' refers to the non-redundant nature of information captured by the defects attributes and their values that are used to classify defects (Butcher, 2002; Linders and Sassenburg, 2004). The chapter 5 explains this concept in more detail.

However, the usefulness of first two models from above has been heavily criticised. In the first place, they assume different way of working and that also does not reflect reality (Fenton & Neil, 1999; Hamlet, 1992; Wood, 1997; Hecht, 1997; Whittaker, 2000; Li, 2003); and as a result, both models can produce dramatically different results for the same data set (Fenton and Pfleeger, 1997; Gokhale, 1996). These are also inadequate for expressing the depth of detail necessary to describe software processes. Also that, these techniques offer a formula, but using them early for cost benefit analysis of SPI techniques is not possible. They also make no attempts to describe the relationship between the variables other than the statistical relationship. These cannot also work with incomplete data when some metrics data is missing. One must wait until later in the life cycle to be able to use them, thus predictions are not available when needed.

Fenton and Neil also note that many reliability models also failed to consider crucial notion of uncertainty. SPI decisions are primarily taken by humans, and therefore the outcome is often uncertain. Uncertainty can be either caused by the unpredictability of future events, or it can also be caused by limitations on the accuracy of the data. Available cost benefit analysis techniques do not include any explicit analysis of risk or uncertainty. They either assume that all future costs and benefits are known or assume that there is no reasonable way to include uncertainty in the calculations. Such
assumptions are obviously unrealistic, and it is clear that decisions would benefit from more explicit consideration of the uncertainties affecting future costs and benefits. Here it is claimed that software uncertainties can be modelled using probabilistic notions of Bayesian Belief Networks, which can provide support for implementing effective software process and can also be used to analyse return on investment for SPI techniques. More about Bayesian Networks is explained in next chapter.

2.5 Software Inspections

During the last decades, many software development organisations have initiated software process improvement programs (Fuggetta, 2000; Humphrey, 2002). The intention of these initiatives is to improve the software manufacturer’s performance by reaching, for instance, the higher levels of process maturity models, such as the Capability Maturity Model or CMM (Paulk, 1993; Software Engineering Institute, 1995), its successor the Capability Maturity Model Integration or CMMI (Chrissies, 2004), and ISO/IEC 15504 in combination with the ISO/IEC 12207 standard (ISO 1995, 2002, 2004). Over the last decade, many improvement models/programs/processes have been proposed. However, because of the time limitation, one single researcher cannot do the analysis for all, therefore in this research, only the cost benefits for Software Inspections process are thoroughly analysed.

We have chosen the Software Inspection process as it is one of the most measured, analysed, and used processes in the history of software engineering. Today, many organisations have made commitments to initiatives in the SEI’s Capability Maturity Model (CMM) in order to deliver good quality products. Humphrey (1989) states that the practice of Software Inspections is associated with Level 3 on the CMM process assessment level. Another reason why the focus is on Inspections is because it is also the most ubiquitous, researched, and reported upon SPI method ever. There is enough data for investigation analyse, and also there is enough experience with the field such that
one can try to tackle more difficult issues that cannot be addressed through other researchers' experiments.

2.5.1 Background and History

Michael Fagan proposed the Software Inspection process in 1976 (Fagan, 1976). This was a formal, well-defined approach to find and correct defects in software. Many studies and success stories have proved the benefits of Inspections since then. With more research being done over time, the original Inspection approach has been refined into many different variants, specifically tailored for different kinds of conditions. Finding out about Inspections has not been easy to reconcile and consolidate due to the sheer volume of work already published. Hence, in this thesis only the most relevant published research is covered.

The IEEE glossary of software engineering provides the following definition of a Software Inspection (IEEE, 1990):

\[ A \text{ formal evaluation technique in which software requirements, design or code are examined in detail by a person or group other than the author detecting faults, violations of development standards and other problems.} \]

Software Inspections have been shown to be a practical process for ensuring that artefacts created throughout the software lifecycle possess the required quality characteristics. For example, Inspections have been used to improve the quality of design-specification by helping to detect and remove defects during design phase. In this way, Inspections help reduce the number of defects in a software system by ensuring that its artefacts correctly reflect the desired quality properties.

Russell (1991) points out that the organisation that adopts Software Inspections benefits by improved predictability in cost and schedule performance, reduced cost of development and maintenance, reduced defects in the field, increased customer
satisfaction, and improved morale among practitioners. In fact, it has been claimed that Inspection technologies can lead to the detection and correction of anywhere between 50 percent and 90 percent of the defects. Early defect detection and removal improve the predictability of software projects and help project managers stay within schedule, since problems are exposed throughout the early development phases.

2.5.2 Integration of Software Inspection in Development Context

Stavely (1999) observes that a defect can be characterised as any product variance; that is, any deviation from the required quality properties. In all software development phases defects are introduced, found and rework is then carried out. However, often most defects are only found when the software product is almost finished, e.g. during the system and acceptance testing phase, or even during operation. Defects found during the testing phase have the disadvantage that the rework on the almost finished software product could be very time consuming. It would have saved the development organisation a lot of time if these defects were found during an earlier development phase. Therefore, one common reason for the use of Inspection technology in software projects is the increased possibility of early predictability of defects.

Inspections are generally accepted as a means to improve the quality of software products in an effective and efficient way. However, Inspections are not a standard practice in a great number of software projects and software organisations. Introducing and implementing Inspections could often be a tedious and difficult task as software engineers must be personally satisfied with the effectiveness of new methods before they will consistently use them.

Literature demonstrates that the use of Inspection is biased towards code documents. However, many papers talk about the Inspection of requirements, design and test-case documents also (Doolan, 1992; Ebenau, 1994; Fagan, 1986). They provide evidence that, although code Inspection improves the code quality and provides savings, the
savings are higher for early life-cycle work-products. The results given by Fagan (1976) reveal that the introduction of code inspection saved 39 percent of defect costs compared to testing alone and the introduction of design inspection saves 54 percent of defect costs compared to testing alone. These findings motivate the use of inspections especially throughout the early development phases.

Inspections are an effective and efficient measure that can be introduced to improve the quality of the products at an early stage. Besides identifying defects at the earliest stages, preventing of defects is the important issue. Inspections can also be used as a means for defect prevention by improving the process for development. Based on an analysis of the defects that were identified, the software development processes can be adapted and optimised to prevent these defects from occurring in the future. Software engineers involved in the inspection process can learn from their defects or the defects that were made by someone else.

There are more than 20 distinct purposes that inspections serve at varying degrees. Such as improving document quality, removing defects, job training, motivation, helping a document author, improving productivity, and reducing maintenance cost (Radice, 2002). Each inspection addresses several of these purposes to varying degrees. Inspections help, but still the principal issue is how to motivate and implement inspections within a software organisation. According to Gilb (1988), metrics should play a major role in convincing both the software engineers and their management, and in tuning the inspection process. In fact, metrics are a critical success factor to successful inspection implementation.

Sections 2.5.3 and 2.5.4 briefly points out inspection process, and inspection roles. The process has been explained in detail in chapter 4, where we also propose Bayesian Belief Network to implement effective inspection process.
2.5.3 Software Inspection Process

Figure 2.1: Software Inspection process (Gilb & Graham:33, 1993)

Figure 2.1 shows the Inspection process described by Gilb and Graham (1993). Following sections briefly elaborate the various steps of the process.

- **Kick-off Meeting**

  Communication is critical to the success of any process (Jones, 1996). One key point in communication is the Kick-off meeting, which helps at the start of the process. The decision to hold a Kick-off meeting depends on whether the Inspection handbook guidelines require such a meeting, or if the Inspection leader judges that it is necessary or advisable due to the special nature of the material or the special nature of the participants, for example, new checkers or outside guest checkers.

- **Individual Checking**

  The time we spend in individual checking is probably the most important in the Inspection process. We must spend enough time so that we will be effective at
finding issues. This involves studying the documents, comparing them against each other, going through rules and the Checklist and so on.

• Logging Meeting

As explained by Gilb and Graham (1993), the logging meeting has three purposes: 1. To record all potential defects identified during individual checking, as issues in the issue log; 2. To perform the checking process in a group environment to identify additional issues, which had not been found during individual checking, so that they are also logged as issues; and 3. To record other items, that is, improvement suggestions and questions of intent to the author.

• Edit and Follow up

The point of product Inspection is to remove defects from the product, and this is done in the edit phase. Gilb and Graham (1993) explain that the Editor is the first person to acknowledge that an issue is really a defect. Generally five people may be involved in this activity, and often with different viewpoints. A single person finally makes satisfactory correction. The correction is then effectively checked in later Inspections and tests.

The Inspection leader performs the follow-up phase, which includes checking that the editing has been completed and that every issue has had some action taken. The Inspection leader is responsible for follow-up but is not responsible for ensuring the correctness of the actions taken – only that everything has been acted on. Gilb and Graham (1993) point that the action may be a correction, a change request, or an inserted written comment to avoid further misunderstandings.

It is even more important that the product when exited is of the quality, which is required, so that the software development process which will take place next on
this product will not waste the developer's time. The Inspection leader and moderator are responsible for checking that all exit criteria have been met.

2.5.4 Inspection Roles

Software Inspections are conducted by a number of people with defined roles: author, moderator, inspectors and scribe.

The author is the individual who produces or modifies the work-product to be inspected. The author can also be referred to as the producer when the work-product is initially produced. The author can help the other participants by focusing their attention on known open issues and problem areas where he or she is concerned or wants special attention paid by the inspectors.

The moderator is responsible for ensuring that Inspection procedures are performed throughout the entire Inspection process. This includes ensuring that the other Inspection team members perform their roles to the best of their ability.

The scribe (or reader) is the inspector who leads the team through the material during the Inspection meeting. The purpose of reading is to focus on the Inspection material and to ensure an orderly flow for the inspectors.

For requirements specifications, the scribe should be reading to ask whether the requirements are complete, correct, consistent, and whether they can be implemented. For design, the scribe should give some interpretation stating how the design could be implemented, since the design must be implementable to satisfy the functional requirements. For code, the scribe not only reads the material, but also interprets the related design and code.
Some Inspections have a separate role of Recorder, who records the data for defects found and data about the conduct of the Inspection.

Regarding Inspection roles, Gilb and Graham (1993) suggest that, in all type of Inspections, it is essential that at least one of the assigned inspectors has expertise in the specific knowledge area so they can focus on the key issues: standards, interfaces, maintainability, usability, complexity and security.

2.5.5 Constraints to achieve 100% Inspection Effectiveness

Even after much research on Inspection process, two questions still surface: 1. Can all projects and all organisations see 100% Inspection Effectiveness? 2. Which projects should be selected to achieve this goal?

Literature does suggest that every organisation should be able to experience closer to 100% defect removal for some Inspections. Radice (2002) gives some examples of 90%+ effectiveness, so we know these numbers are achievable, but why not within all organisations? The reason must be the existing variability in all processes. One of the factors that leads to variability is people. Each person has different experiences, skills, knowledge and capabilities. We cannot assume all people are equally capable even when we provide them the same training, processes, tools, and so on. However, we can bring them all into a higher range of capability. Some other examples of variables affecting effectiveness are the quality, complexity and size of the work-product being inspected. This research tries to consider all these factors to achieve higher effectiveness of Inspection processes.
2.5.6 Cost Benefit Analysis for Software Inspections

Benefits of following Inspections include reduced development and maintenance costs, improved customer satisfaction, increased profitability; additional sales because of improved quality etc. While Inspections are highly cost-effective, they are also labour intensive. While implementing Inspections, we must estimate and measure the cost of effort invested and spent in Inspections. Costs of implementing Inspections include the costs of establishing technical infrastructure to support the process improvement activities, training and start up cost for kick-off sessions, causal analysis meeting, action team meeting; and documentation and database costs etc.

Most of the stated benefits and costs associated with Inspections have already been demonstrated in many projects and organisations. However, most of the published Inspection work has been integrated into a broader context, that is, into a large body of knowledge, hence making the work difficult to evaluate for software practitioners. To provide a systematic view of the research and practice in Software Inspections, we developed a model from available results in accessible Software Inspection publications.

There is vast amount of literature which contributes to the knowledge of Software Inspection by identifying factors that may impact Inspection success. However, none of them present their findings from a global perspective. This makes it difficult for practitioners to determine where to concentrate if they want to introduce Inspections or improve on their current Inspection approach. Even though there is a great deal of material available, these all definitely needs merging, as several controlled experimental studies independently done by Fagan (1976), Jones (1991), Shirey (1992) and Votta (1993) still cannot answer all of the following questions: Does every Inspection need a meeting? What is the most effective size of an Inspection team? What is the most effective reading rate for an Inspection?
If our developed model can be used effectively to meet organisational needs, then it has capability to answer all of these questions. Researchers and practitioners can profit from this research in different ways. They can use the presented model to identify fruitful areas for future improvement. They can find a road map that would help them to focus quickly on the most cost effective Inspection approach adapted to their particular environment.

2.6 Problems with existing Cost Benefit Analysis Methods

Cost benefit Analysis is an art consisting of a series of techniques useful for decision making (Dively & Zerbe, 1999). The cost benefit analysis of SPI helps to assess the profitability of implementing the process and evaluating the input against the output in monetary terms. It can be also used to determine how much money is lost from creating and using a new and improved software process.

In his book, Rico (2004) explains techniques to do cost benefit analysis for different SPI techniques; such as Benefit/Cost ratio (B/CR), Return On Investment (ROI) and Net Present Value (NPV); with different SPI models and strategies such as Software Inspections, Personal Software Process (PSP), Team Software Process (TSP), Capability Maturity Model (CMM), ISO 9001 and Capability Maturity Model Integration (CMMI). O’Neil (2005) has demonstrated the calculation of return on investment using Net Present Value (NPV) formula.

In our context, benefit is the economic value resulting from a new and improved software process. Cost is the amount of money you must spend to get something back. Benefit/cost ration (B/CR) is simply the ratio of benefits to costs. B/CR is a measure of how much money is gained from using a SPI method. For example, a B/CR ration of 2:1 means that for every pound we spent, two pounds is returned.
ROI (Return on Investment) is the amount of money that is gained after spending an amount of money. That is, ROI refers to the amount of money gained. For example, an ROI of 10% means that for every pound you invest, 10 cent is returned.

\[ \text{ROI} = \frac{\text{Benefit} - \text{Cost}}{\text{Cost}} \times 100\% \]

Net Present Value (NPV) is what money is worth in the future. NPV is the economic value of today’s money in the future less inflation. For example, if we assume inflation rate of 5% per year, £10 today will be worth £9.52 a year from now.

\[ \text{NPV} = \frac{\text{Benefit}}{(1 + \text{InflationRate})^\text{year}} \]

At first, these techniques look quite easy, essential and simplistic in the field of software process improvement as they do not involve more than one or two significant terms or inputs; and also support core of cost benefit analysis that benefits do exist, and costs must be counted.

However, these techniques might not be good for a variety of reasons. Firstly, none of them express the depth of detail necessary to describe software processes. These techniques offer a formula, but using them early for cost benefit analysis of SPI techniques is not possible. They also make no attempts to describe the relationship between the variables other than the statistical relationship. They cannot also work with incomplete data when some metrics data is missing. One must wait until later in the life cycle to be able to use them, thus predictions not available when needed.

Also that, SPI decisions are primarily taken by humans, and therefore the outcome is often uncertain. Uncertainty arises in many situations. For example, experts may be
uncertain about their own knowledge. Uncertainty is a result of either the unpredictability of future events, or by limitations on the accuracy of the data. Available cost benefit analysis techniques do not include any explicit analysis of risk or uncertainty. They either assume that all future costs and benefits are known or assume that there is no reasonable way to include uncertainty in the calculations. Such assumptions are obviously unrealistic, and it is clear that decisions would benefit from more explicit consideration of the uncertainties affecting future costs and benefits.

As explained earlier organisations want to improve their existing processes, but they do not know exactly what to change. They cannot predict with certainty what will be the future output and how many defects the work-product will have after following certain processes. It is because they are not sure which inputs that are not yet executed, would produce a failure if executed. Had they known this, they could use the information for defect prevention or process improvement.

There is necessity of the approach to recognise the uncertainty and factor it into the cost benefit analysis, which clearly offers following advantages:

- The explicit recognition of uncertainties helps decision makers understand the quality of process used to support a particular decision and gives them an idea of potential problems in the analysis.
- Analysing the impacts of uncertainty on a cost benefit analysis often highlights factors for which better information is needed.
- This also reveals factors that have the greatest influence on the possible results of the project. Once these factors are recognised, it may be possible to modify them to get maximum return on investment.
- If uncertainties are analysed in the preliminary stage of the project, it is possible to suggest conditions that indicate when particular process should be terminated

Constructing a graph representing causal relations between events of the process can only capture this uncertainty (Jensen, 2001). As explained earlier, here we use Graphical Probability Networks, also known as Bayesian Belief Networks, to capture this
uncertainty; which not only provide support for implementing effective software process, but also helps to analyse return on investment for SPI techniques. More about Bayesian Networks is explained in next chapter.

2.7 Available Modelling Approaches

A number of alternative approaches to modelling relationships have been developed, e.g. Non-monotonic reasoning, Fuzzy logic and Artificial Neural Networks. Out of these all possibilities, we have to choose an approach that can help to model the effectiveness of software processes that makes use of previous experience and expert judgement.

Non-monotonic reasoning (Reiter, 1987) is an approach to learning where knowledge is logically ordered to provide diagnosis of problems. Inference is obtained by applying a set of rules to determine the logic pathway to the resolution of the problem. The result is deterministically based on a rule base and has no ability to learn from the actual outcome of the event. The approach is qualitative rather than quantitative, and was therefore rejected.

Fuzzy Logic (Zadeh, 1983) in this context extended the application of non-monotonic reasoning by applying overlapping boundaries to decision points in the set of logical rules. However, a fuzzy model approach cannot be used for diagnosis, as basically it cannot adopt its rule base on the actual outcome.

An alternative approach would have been to use an Artificial Neural Network to create a model (Czachur, 1995). The major disadvantage with this type of approach is that it requires a very large amount of data compared with a Bayesian Belief network to establish a model. As discussed previously, one of the key issues in determining the effectiveness of SPI techniques is team experience. The selection of Bayesian Belief
Network (BBN) method provides a means for including this as the prior probabilities within the network.

We believe that Bayesian Belief Networks (also known as Belief Network, Causal Probabilistic Network, Causal Net, Graphical Probability Network and Probabilistic Cause-Effect Model) are one of the suitable methods for modelling software processes in the context of problems identified from current literature. We need such comprehensive models, so that we can measure, control, predict and improve software quality by taking into account both historical data and experts' knowledge (Fenton, Krause and Neil; 2001).

Neapolitan (1990) explains that Bayesian Belief Networks are very effective for modelling situations where some information is already known and incoming data is uncertain or partially unavailable, unlike rule-based expert systems where uncertain or unavailable data results in ineffective or inaccurate reasoning. These networks also offer consistent semantics for representing causes via a sensitive graphical representation. Because of these all capabilities, Bayesian Belief Networks are being increasingly used in a wide variety of domains where automated reasoning is needed. In the software industry, BBNs are already used for software debugging, printer troubleshooting, safety and risk evaluation of complex systems, help facilities in Microsoft office products and so on.

Next chapter describes concepts and usefulness of BBNs in more detail.
Chapter 3

Bayesian Belief Networks

Abstract

This chapter makes the case for the use of a Bayesian technique to address SPI problems. The chapter first outlines the essential features of Bayesian networks. At the end of the chapter, we also discuss an example (using an Influence Diagram), which briefly explains how we can use the Bayesian technique to analyse costs and benefits of SPIs.

3.1 Introduction

In real world, the degree of belief about an event is many times uncertain; and sometimes uncertainty is unavoidable because events in real-life could be quite ambiguous and change frequently. For example, our belief about whether it will be raining tomorrow is uncertain if we do not know about the weather forecast. Although the uncertainty is ambiguous, it also represents information about our belief and the level of our knowledge. The modelling of Bayesian networks which uses probability theory and graphical representation has provided an efficient way to deal with uncertainty.

Since the early 1990s, Bayesian Networks have attracted a great deal of attention among researchers in Artificial Intelligence (AI). Fenton, Krause and Neil (2001) also points out that a Bayesian Belief Network is a relatively new but rapidly rising technology, which has provided an elegant solution enabling us to solve many problems - mainly
related to uncertainty. Bayesian Belief Networks are powerful tools for modelling causes and effects in a wide variety of domains. They are compact networks of probabilities that capture the probabilistic relationship between variables and historical information about their relationships, and thus enable reasoning under uncertainty.

The following sections describe a complete overview about Bayesian networks’ probability theory and graphical representation.

### 3.2 Graphical Networks

A graphical probabilistic network is a network that represents a problem domain of a system for reasoning under uncertainty. The basic concept of a graphical probabilistic network is it provides information on some events which influence your belief of other events. The dependencies among events are represented by a graph. In the graphical network, the events are random variables which are represented by nodes. The relationship between nodes is called a causal relation which is indicated with a directed edge. An edge from a to b indicates that a has impact on b or we say that a is the parent of b, and b is the child of a.

![Graphical Network](image)

**Figure 3.1:** Graphical Network
3.2.1 Causal Networks

The dependencies between events and beliefs of events can be illustrated on the following example using graphical networks (Jensen and Lauritzen, 2001).

Wet Grass Example

In the morning when Mr. Holmes realises that his grass is wet. He wonders whether it has rained (Rain) during the night or whether he has forgotten to turn off his sprinkler (Sprinkler). He looks at the grass of his neighbours, Dr. Watson to find out if it has been raining (Jensen and Lauritzen, 2001).

The situation above can be represented by a graphical network or causal network (figure 3.2). A causal network is a graphical model which consists of variables and directed links between variables. Causal networks provide an intuitive graphical visualisation for the problem's knowledge under uncertainty. Based on expert judgement on the problems of wet grass, we can decide that the relevant events or variables of the wet grass problem domains are Rain, Sprinkler, Watson and Holmes. The links between the events tell us about causal relations among them — that, event Rain is the 'parent' for events Watson and Holmes and Holmes is the 'child' of event Sprinkler.

Figure 3.2 is a graphical network that illustrates the situation of the wet grass problem. Based on the observation of the grass is wet, there are several possibilities: whether it rained the previous night or Mr. Holmes had forgotten to turn off the sprinkler. With no further information, his belief on both events of sprinkler is on and it has been raining are increased. However, if Mr. Holmes looks at the grass of his neighbour Dr. Watson and finds out that his grass is wet then this further information increases his belief that it has been raining and decreases his belief that sprinkler is on. As shown in figure 3.2, the uncertainty or certainty of an event can affect our belief on other events. In addition, the relationships among these events are changed if the certainties of belief are changed. In
later sections, the further explanation about the reasoning of changing of belief in causal networks is given.

Figure 3.2: The Causal Network for Wet Grass Situation

3.3 Connection Patterns

The interpretation of a graphical network always describes the connection or influential patterns in the network. The pattern of connections can be interpreted in terms of causation - for example, if X & Y are dependent and conditional on an effect Z, it is difficult to discover if X and Y are independent. On the other hand, if Z is dependent on the effect X and Y, then X and Y clearly are independent to each other. From the graph in figure 3.2, we can observe the influence among the events for the problem domain of wet grass in more details. In the following sections, we will expand the wet grass network to include the effects of raining and explain the causal dependencies patterns in graphical networks.

3.3.1 Serial Connection

In figure 3.3 the network shows the dependencies of serial connection in the pattern of \( X \rightarrow Y \rightarrow Z \).
Based on the graph pattern, we say X has influence on Y which has influence on Z. We believe that information may be transmitted from X to Y to Z unless the state of Y is known. This also means that X and Z are independent if and only if the state of Y is known.

Now let us see the serial connection portrayed in wet grass situation. We believe that a dark cloud might cause raining, and we know rain may make the lawn wet.

\[
\text{dark cloud} \rightarrow \text{rain} \rightarrow \text{wet lawn}
\]

However, if we know that it has been raining, (the state of rain event is known) then the events of dark cloud has no impact to our belief about the wet lawn and vice versa.

\[
\text{dark cloud} \rightarrow \textbf{rain} \rightarrow \text{wet lawn}
\]
3.3.2 Diverging Connection

The graph below shows the diverging connection with the pattern $X \leftarrow Y \rightarrow Z$.

![Figure 3.4: Diverging Connection Pattern](image)

From the influential point of view, we know that $Y$ has causal influence to both $X$ and $Z$. The influence relationship can also be noticed by the edges between the nodes, where there exist edges from $Y$ to $X$ and from $Y$ to $Z$. From the connection pattern, we know that $X$ and $Z$ is conditional independent if and only if the state of $Y$ is known.

To put the diverging connection into the situation of wet grass, we can say that rain caused Watson’s grass to be wet and also caused Holmes’s grass to be wet.

![Box: Watson ← rain → Holmes](image)

However, if we know that it has been raining, then the information about Watson’s grass is unrelated to our belief about Holmes’s grass is wet. On the other hand, if we know nothing about the state of rain, if Watson’s grass is wet this will lead our belief that Holmes’s grass is also wet. This is because both events of Watson and Holmes are independent when there is no information about raining.

![Box: Watson ← rain → Holmes](image)
3.3.3 Converging Connection

In the converging connection of figure 3.5, the graph structure indicates both X and Y has influence on Z. Information may only flow through converging connection if the state of Z is known or either the state of X or Y is known. That is, X and Y are independent if and only if the state of Z is not known.

![Figure 3.5: Converging Connection Pattern](image)

Now we put the converging connection into the wet grass situation. The grass is wet state may be because the sprinkler was on or it had been raining.

\[
\text{sprinkler} \rightarrow \text{wet grass} \leftarrow \text{rain}
\]

If we checked that the sprinkler is turned on, then this reduces our belief that it has not been raining. Therefore, if we know that the grass is wet and the sprinkler is turned on, then our belief that the grass is wet because the sprinkler is on is increased and our belief that it has been raining is decreased.

\[
\text{sprinkler} \rightarrow \text{wet grass} \leftarrow \text{rain}
\]
3.4 d-Separation

Information or evidence that we know or we do not know about an event is very important in causal network as it represents the conditional independence of other variables in the network. The rule for deciding how information or evidence is to be transmitted in connection patterns of serial connection, diverging connection and converging connection is known as d-separation. As we have discussed, in the above three connection patterns, the dependency of two variables in causal networks is dependent on how much information we know about a third variable. The rules of d-separation clarify the causality of variables in terms their dependency. The following are the rules for d-separation by Jensen (1996).

Variables A and B in causal a network are d-separated if for all paths between A and B there is a third intermediate variable Z that either

- The connection is serial or diverging and Z is instantiated or
- The connection is converging and neither Z nor any of its descendants have received evidence

Thus, the causal relationships between variable A and variable B is probabilistically independent of conditioning on variable Z which blocks the path between them. The purpose of d-separation is to induce the causal directionally for statistical prediction in causal network, which in turn clarify human reasoning of uncertainty (Jensen, 1996). For example, if X and Y are d-separated by Z, then our certainty about X has no impact on the certainty of Y when we know about Z (Pearl, 1988).

While modelling a Bayesian Network, we should ensure that the model is validated as per d-separation properties. Following example (figure 3.6) shows the causal relations among Salmonella infection, flue, nausea, and pallor (Jensen, 2001).
Knowledge of one possible cause of an event does not tell us anything about other possible causes. However, if anything is known about the consequences, then information on one possible cause may tell us something about the other causes. If we know nothing of nausea or pallor, then the information on whether the person has a Salmonella infection will not tell us anything about flu. It means that Salmonella and Flu are separated if we do not have knowledge about Nausea. However, if we noticed that the person has Pallor, then the information that he does not have a Salmonella infection will perhaps incline us towards believing that he has the flu.

3.5 Bayesian Networks Definition and Notation

Bayesian Networks is formalism for reasoning under uncertainty (Pearl, 1988). Probability theory and graph theory are basics of Bayesian networks for representing a problem domain. In Bayesian networks, a problem domain is represented by relations among domain variables and the relations are determined by conditional probabilities. Therefore, a Bayesian networks consists of two parts, a qualitative part and a
quantitative part. The qualitative part is a graph of entities that represent random variables. The graph structure is a form of directed acyclic graph consisting of a set of nodes and a set of edges to represent a problem domain. Bayesian network uses the probability theory of conditional joint distribution to represent the relationship between variables. The value of the joint distribution specifies the strength of relations between the variables. The conditional joint distribution for each variable in the network is represented in a conditional probability table.

Bayesian network uses the probability theory to represent the inference of conditional dependencies and allows decision making under uncertain conditions. In addition, it is also a subjective probability that expresses a person’s degree of belief in the proposition or occurrence of an event based on the person’s current information (Henrion et al., 1991). In terms of theoretical definition, a Bayesian network $N$, consists of graphical model $G$, and probabilistic distribution $P$ where

$$N = (G, P)$$  \hspace{1cm} (1)$$

In the Graphical model, $G$ represents the set of nodes and $V$ represents the set of arcs,

In the graphical model, $G$ represents the set of nodes and $V$ represents the set of arcs, $E \subseteq V \times V$, which links between the nodes. The set of node with the arcs from the direct acyclic graph,

$$G = (V, E)$$  \hspace{1cm} (2)$$

Each $x$ node, in the graph is attached with parents, $pa(x)$. So the conditional probability distribution is:

$$P = \{ p(x \mid pa(x)) \}, x \in V$$  \hspace{1cm} (3)$$
The conditional probability specifies the strength of relation between the child and the parent. The joint probability distribution of Bayesian network over the node \( x \) can be given by the factorisation:

\[
p(V) = \prod_{x \in V} p(x \mid pa(x))
\]  

3.5.1 The Chain Rule

The chain rule establishes that Bayesian network is a representation of a join probability distribution. Suppose we have a domain \( U \) of \( n \) variables, \( X_1, \ldots, X_n \), we have a universe of variables \( U = \{X_1, \ldots, X_n\} \). Now we compute the joint probability distribution \( p(U) \) over \( U \) using the chain rule:

\[
p(U) = \prod_{i=1}^{n} P(X_i \mid X_1, \ldots, X_{i-1})
\]  

\[
= \prod_{i=1}^{n} P(X_i \mid pa(X_i))
\]

Given the joint probability distribution over a set of variables, the structure of a Bayesian network can be reflected by a causal graph. With the graph and understanding of probability distribution, a problem domain can therefore be determined. However, there is always uncertainty in Bayesian network about the degree of belief. In order to deal with uncertainty, Bayesian probability network collects evidence about the domain problem and update the probabilities of the event. The process of accumulating evidence for changing the probability of a Bayesian network is called inference.
3.5.2 Bayesian Network Inference

The purpose of the Bayesian Network inference is to allow certain reasoning on the degree of belief based on the observed evidence. As the evidence accumulates, the degree of belief will eventually change; and with enough evidence, the degree of belief will be very high or very low. The inference of Bayesian network is based on Bayes’ theorem:

\[
P(A \mid B) = \frac{P(B \mid A)P(A)}{P(B)}
\]  

(6)

Based on the formula, we say that the probability of A given B equals to the probability of B given A times probability of A, divided by the probability of B.

Dealing with the uncertainty in the context of an ongoing process of data collection can be simply stated as, ‘revising current beliefs in the light of new information’ (Sander and Badoux, 1991). Therefore, we can adjust our belief on a hypothesis \( H \) in the light of new evidence \( E \) by computation using Bayes’ rule:

\[
P(H \mid E) = \frac{P(E \mid H)P(H)}{P(E)}
\]  

(7)

H represents a Hypothesis
E represents the observed Evidence

The inference of Bayesian network involves the calculation of:

- likelihood \( P(E \mid H) \), is the conditional probability of observed evidence given the hypothesis
- prior probability \( P(H) \), is the degree of belief about the hypothesis in the absence of evidence
- marginal probability \( P(E) \), is the probability of evidence given no information
• posterior probabilities $P(H|E)$, is the output we are interested where we are interested to know, the probability for the hypothesis $H$ given evidence $E$.

According to Pearl (2001), Bayesian inference is a powerful method and its formalism provides powerful means for learning and data analysis. It can also be viewed as providing domains in which dependencies among variables are known. More detail about Bayes' rule and Bayesian inference is given in Pearl (2001).

The tree structures for BBNs have been developed by Lauritzen and Spiegelhalter (Lauritzen & Spiegelhalter, 1988). Their approach is to define a causal network with links between nodes, in a similar way to the Bayes trees described by Pearl (Pearl, 1988). The network is not limited to the tree structures as cross-links are allowed, provided they do not create a directed cycle.

Their approach is to use the topology of the graph to develop a simplified set of equations to perform local computations. The network example is given in figure 3.7.

![Figure 3.7: A Causal Network](image-url)
The first stage is to build up the evidence potentials within the network. This is done by considering an undirected graph that is formed by providing links between un-joined parents of a common child and by removing the causation direction arrows from the graph. The next stage is to triangulate the graph, this means that there will be no cycles of more than four or more nodes without undirected links (figure 3.8).

![A Causal Network](image)

**Figure 3.8: A Causal Network**

This approach is used to build cliques of the triangulated graph in above graph. Any joint distribution involving nodes on the network can be expressed as a simple function of the individual marginal distribution on the cliques. Therefore, marginal distribution only involve a subset of the nodes on the graph.

In the above example, the cliques are represented by:

1. C1, C2, C3, C4, B1
2. C5, C6, B2
3. B1, B2, A1
The joint probability distribution of the graph is:

\[
\]

However, a computationally simple form can be made if it is represented in terms of the evidence potentials where \(\Psi\) is the evidence potential function.

\[
P = \Psi(C1, C2, C3, C4, B1) \Psi(C5, C6, B2) \Psi(B1, B2, A1)
\]

The evidence potential \(\Psi\) is the function of the conditional probabilities of the nodes on the each clique, which can be represented as

\[
\begin{align*}
\Psi(C1, C2, C3, C4, B1) &= P(C1) P(C2) P(C3) P(C4) P(B1 \mid C1 \& C2 \& C3 \& C4) \\
\Psi(C5, C6, B2) &= P(C5) P(C6) P(B2 \mid C5 \& C6) \\
\Psi(B1, B2, A1) &= P(B1) P(B2) P(A1 \mid B1 \& B2)
\end{align*}
\]

The resulting calculations make it easier to update the belief in the node states, by absorbing evidence into the network using the clique calculations (Spielgelhalter & Lauritzen, 1990).

### 3.5.3 Model Specification

A Bayesian Network consists of a set of variables, a set of edges connecting among the variables and a set of conditional distributions. As described above, in terms of graphical representation, a Bayesian networks is a DAG with nodes that represents
variables and arcs that represent the conditional dependency relations among the variables; and the conditional relationships between the nodes are described by joint conditional distribution using Bayesian probabilistic theory.

The Bayesian network as a DAG provides an intuitive graphical visualisation of a domain model for the problem’s knowledge. The following example illustrates a simple yet typical Bayesian network from Pearl (1988).

![Figure 3.9: A Bayesian Network Representing Causal Influence](image)

**Figure 3.9: A Bayesian Network Representing Causal Influence**

Suppose someone is walking across a lawn and trips down on the grass. Figure 3.9 represents the uncertainty knowledge that we might use to reason about the cause for the event. Bayesian networks describe the possibility of an event given that certain events occurred. The information of the occurred even influences our belief of other events.

**Nodes**

A node represents the variables that we use to describe the events for its reasoning. In the slippery model, the situation is modelled with five nodes: Season, Sprinkle, Rain, Wet and Slippery.
Figure 3.10: Nodes Represent Random Variables in BBNs

Edges

The nodes are then related by edges that link the cause of an event to the effect of an event. Therefore, based on connections and relations of nodes in the graph, we can understand the problem domain by interpreting the links of nodes. The links between nodes represent causal relationships and causal impact among the variables. The presence of an edge between two nodes of a Bayesian networks implies that there is a conditional dependency regarding these nodes.

Figure 3.11: Edges Represent the Causal Relations in BBNs

The edges also describe the parent child relationship in Bayesian networks. When two nodes are linked by edge, it indicates causal dependencies between the variables and reflects cause-effect relations in term of conditional probabilities. It is being drawn from the causative node which is the parent node to its immediate effect node, which is the child node. The child node is usually conditionally dependent on the parent node. A
more detailed description about causal and conditional dependency may be found in Pearl (1988) and Lauritzen et al (1988).

For example in figure 3.12, a likely cause of slippery grass in that the grass is wet, where ‘wet’ is the cause that has direct effect to ‘slippery’. On the other hand, the grass could be wet because either the sprinkler is on or it is raining. Now if we know that it is the rainy season, the sprinkler would not be in use. Therefore, information about the sprinkler has no influence on our belief about the wet grass. We believe it must have rained. However, if we observed that the sprinkler is on, then the grass is probably wet and we infer that it probably did not rain. We reduce the likelihood that it is raining. This is called explaining away (Jensen, 2001).

The example describes the causal relationships among the season of the year, whether it's raining, whether the sprinkler is on, whether the grass is wet, and whether the grass
is slippery. For a DAG, we must specify the conditional probability distribution for each node. If the variables are discrete, this can be represented as a conditional probability table (CPT), which lists the probability of each node that apply causal independent among the variables (Charniak, 1991).

Table 3.1 shows the conditional probability table for Season node which is a parent node in the network. The probability show is the prior probability of Season which is the degree of belief of Season event with no information about other event.

<table>
<thead>
<tr>
<th></th>
<th>Yes</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Season</td>
<td>0.8</td>
<td>0.2</td>
</tr>
</tbody>
</table>

**Table 3.1: NPT for ‘Season’ Node**

On the other hand, table 3.2 shows the conditional probability of event Sprinkler given event Season. The event Sprinkler is the child node of Season, knowing the information about Season will affect the belief of Sprinkler.

<table>
<thead>
<tr>
<th></th>
<th>Yes</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sprinkler</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Yes</td>
<td>0.3</td>
<td>0.8</td>
</tr>
<tr>
<td>No</td>
<td>0.7</td>
<td>0.2</td>
</tr>
</tbody>
</table>

**Table 3.2: NPT for ‘Sprinkler’ Node**
3.6 Influence Diagrams

An enhancement to using a pure Bayesian Belief Network, which just uses basic nodes (also known as chance nodes), is to use utility theory together with an Influence Diagram of the type explained by Jensen (Jensen, 2001). An influence diagram is a Bayesian network augmented with decision nodes and utility nodes (Jensen, 2001). Influence diagrams extend Bayesian probability networks for solving decision making problems by introducing reasoning about decisions. An influence diagram is used as an alternative to a Bayesian networks when there is a problem involving decision making. The decision function of an influence diagram supports reasoning under uncertainty and provides further information to the networks. In addition, it also offers a means to compare alternatives and preferences to decision problems.

The purpose of an influence diagram is to model a decision making situation in an efficient and effective way. The graphical representation of an influence diagram clearly lays out the actions and goals that help decision makers analyse decisions. It captures elements of a problem and the structure of a decision.

3.6.1 Influence Diagram Specification

An influence diagram consists of a directed acyclic graph over chance nodes, decision nodes and utility nodes. The modelling of decision making using influence diagram provides an understandable and clear graphical representation of a decision making problem. The graphical representation of an influence diagram illustrates a decision making situation in an intuitive way which shows the relationships and dependence relations that influence one another. The following figure shows the basic model specification of an influence diagram built using the Hugin software.
Figure 3.13: Basic Nodes of Influence Diagram

- **Chance node** is similar to a random variable in Bayesian networks. It represents the uncertainty situation in a decision making problem.

- **Decision nodes** represent a set of possible actions that we can take control as a decision maker.

- **A utility node** contains values regarding the utility for each action in order to assess the usefulness of the action. A table is associated with utility node where the values are affected by the parent nodes.

Although an influence diagram is an extension to a Bayesian network, there is syntax applied in the graphical structural specification and quantification of an influence diagram. For modelling graphical specification, all the decision nodes are linked by a direct path and utility nodes have no children. For the quantitative specification in influence diagram, the chance nodes and decision nodes have a set of mutually exclusive states. On the other hand, the utility node $U$ has no states but it is attached to a
real-value function over $U$. As for the chance nodes, it is similar to random variables in Bayesian networks where there is a conditional probability table associated with each node. In an Influence Diagram, there must be an unambiguous order among the decision nodes, i.e. there can be only one sequence in which decisions are made.

Jensen (2001) describes the general situation with one decision variable (figure 3.14). There is a Bayesian network structure with chance nodes and directed links. The network is expected with a single decision node $D$. $D$ may have an impact on the structure. In other words, there may be a link from $D$ to some chance nodes. Furthermore, there is a set of utility functions, $U_1, \ldots, U_n$ over domains $X_1, \ldots, X_n$. Graphically, a utility function is represented as a diamond-shaped node with incoming links from the nodes in its domain.

![Connections of Influence Diagram](image)

**Figure 3.14: Connections of Influence Diagram**

The task is to determine the action that yields the highest expected utility, that is, with evidence $e$ achieved, we calculate
\[ EU(D \mid e) = \sum_{X_i} U_i(X_i)P_i(X_i \mid D, e) + \ldots + \sum_{X_n} U_n(X_n)P_n(X_n \mid D, e) \]

and a state maximising \( EU(D \mid e) \) is chosen as an optimal action.

The above equation only considers situations where we request posterior probabilities for single decision variable. This may not be sufficient because \( X_i \) may contain more than one variable.

Figure 3.15 shows a typical example of an influence diagram. It is a decision making situation where we have an opportunity to decide whether to carry an umbrella (Jensen, 2001). The purpose of the influence diagram is to help us make the most effective decision based on the options given. In figure 3.15, the chance nodes of Forecast and Weather are used to show probabilistic information about the possibility of ‘it is going to be raining’ or ‘no rain’. The action of whether to carry an umbrella is in the decision node of Umbrella with two states of ‘Take’ or ‘Do Not Take’. However, the decision making can be evaluated using the utility node. The utility node of ‘Satisfaction’ shows
the value of each action to be taken. The link from decision node to utility node means the actions influence the level of our satisfaction with our action.

If we have the information for both Forecast and Weather that is going to rain (figure 3.16), then the utility node of Satisfaction show our satisfaction value of 100 if we carry umbrella with us. On the other hand, if we do not carry the umbrella and it starts raining, then the value of our satisfaction is 0.

![Influence Diagram Example – with evidence](image)

**Figure 3.16:** Influence Diagram Example – with evidence

### 3.7 Using Influence Diagram for Process Improvement

Influence Diagrams have applications where there is a need to know the influence of variables in a complex relationship in determining whether to follow a specific course of action, or not. The example shown in figure 3.17 should explain why we use application of Influence Diagram to analyse costs and benefits associated with process improvement.
In the example, Quality of work product depends on Quality of Requirements defined and achieved CMM maturity level. The CMM (Capability Maturity Model) is a set of guidelines, which has five certification levels of software maturity. The Software Engineering Institute (SEI) asserts that these levels capture relevant company processes, and lead to desired product quality. The CMM says that applying several significant (and often expensive and time-consuming) changes causes improvement. These changes progressively move the company from a low maturity level to a higher level; and it is assumed that higher the CMM maturity level, better the quality of work product. Quality of work product also depends on the quality of Requirements defined. The relation among these variables (Requirements, CMM and Quality nodes) is shown in figure 3.17.

Now the company may decide to invest resources in giving some training to team members to improve the Requirements defined, and thus to improve the overall Quality. Since this involves a decision through period, we have added three nodes very similar to those already in the network. The new nodes Requirements', CMM' and Quality' represent the same as the old nodes, except that they represent the state after the training given. The diagram also shows the causal dependence between old and new nodes. The action of investing for Training is added as a decision node Training. We have given the link from Training to Requirements', as we expect that the training to have an impact on
the future quality of Requirements. To measure the utility of the decision, we have added utility nodes Cost and Benefit to the diagram, each contributing with one part of the total utility. The utility node Cost represents the information about the cost of training given, which the node Benefits represents the utility achieved at the end. To get the quantitative representation of the Influence Diagram, we also constructed assumed probability tables for each node. Figure 3.18 shows the complete representation of the Influence Diagram.

![Influence Diagram](image)

**Figure 3.18:** Using Influence Diagram for Process Improvement – with NPT

The example shows that Influence Diagram can highlight the decision, which maximises the expected utility. The company would decide to spend on training if the expected Benefits after the training are more than the Cost of the training. It should be noted that there are certain costs with SPI methods like CMM. However, they do not guarantee benefits. What to do if cost is 70 units and benefits are only 60 units? In these scenarios it is better to invest on something else, than to achieve higher maturity level. The CMM and similar frameworks help to assess the quality aspects of software processes, but companies cannot certainly base their improvement strategies only on them. The other
general problem with all improvement initiatives is that the uncertain gain lies in the future. In actual conditions, it is very difficult to make the decision under this uncertainty. Thus, the example points out that the Influence Diagram theory can efficiently provide the decision-maker with the data to make decisions.

In comparisons of other approaches we use BBNs to analyse SPI cost benefits, as they have the ability to model problems through the application of causal networks providing a structure and relations for the influence factors. They are also unique in being able to make sensible predictions without having complete dependence on the past data. In the very first processes, organisations have no formal knowledge. Knowledge is not held in databases but is the total sum of people’s experiences. This knowledge is valuable but because it is intangible is often too easily ignored. Measurements for process improvements should be done not only on basis of any available past data but also on the experts’ knowledge. We believe that BBNs are one of the best methods for modelling, measuring and controlling process improvement.

In the next chapter, we explain development of such BBN, which can be used to provide what-if analyses and a range of predictions for implementing effective Software Inspection process.
Chapter 4

BBN for Inspection Effectiveness

Abstract

This chapter gives detailed explanation of the development of a Bayesian Belief Network for measuring the Software Inspections Effectiveness, which mainly considers how different factors affecting the Inspection process are related. The model described adequately answers the basic hypothesis set out in chapter 1, and also lays the foundation for Chapter 5.

4.1 Introduction

Madachy, Little and Fan (1993) described Inspection effectiveness as defects found per unit of Inspection effort. According to Gilb and Graham (1993), effectiveness is the percentage of total major faults or issues found in Inspections.

Inspection effectiveness varies significantly across organisations or, even more striking, from one Inspection to the other in a given organisation. To find the effectiveness of Inspection process, project managers sometimes only observe the cost of Inspections and usually ignore the benefits of achieving higher product quality. It is necessary to implement effective Software Inspection process to gain maximum benefits from it. Cost benefit analysis of Software Inspections can be obtained by calculating an estimate of its effectiveness.
The software engineering community has identified a number of factors that have an impact on Software Inspections performance. For example, effect of reading technique used (Porter and Votta, 1994), the process of Inspections (Gilb and Graham, 1993; Madachy et al, 1993; Porter et al, 1995), the explicit criteria for deciding when to stop Inspections (Porter et al, 1995), and so on. Many researchers (Cockram, 2001; Jones, 1991; Votta, 1993) present models, some with proposals for improvements in effectiveness. However, they all have in common is that they only map a fraction of the Inspection process to a model, regarding only one or two aspects, e.g. the influence of team size and/or reading technique on the Inspection process efficiency.

While today many factors are known to influence the Inspection process, not all of them are understood very well. Furthermore, they form a complex network of interrelations, with one factor influencing many others and being influenced by many others. A model that comprehends all or at least the most important influence factors would allow simulating the complete Inspection process and drawing conclusions from this.

The application of proposed Software Inspections effectiveness model provides industry with a more efficient way of conducting Inspections, and in concentrating on the more important attributes of an Inspection to gain more benefits from invested efforts. The effectiveness model can be used for planning purposes; for example, to estimate the preparation effort before and after the Inspection is conducted; or for control purposes, for example, to achieve a higher level of work product quality.

While developing the model, we first tried to identify influence factors from the literature. Then, we identified individual factor's relationship with other factors. As can clearly be seen, the influence relations of the factors do not have a tree-like structure, but form a network of interrelations (Appendix B). Many factors influence not only one process step, but two or three. This must be considered when trying to tune single phases: to ensure the improvements made in one step do not lead to deteriorations in later steps.
4.2 Model Structure and Influence Factors

The Inspection based process model is mainly based upon an extensive literature study. Having identified the variables for the model, we developed the Bayesian Belief Network using AgenaRisk (Agena, 2005). This choice was the result of a fairly comprehensive comparison with other tools such as HUGIN (Hugin, 2005) and Netica (Norsys, 2005). We preferred tool AgenaRisk mainly because we wanted to use features such as Expressions & Partial expressions to fill the initial NPT tables.

To decide the model structure, we do not have a formal method. It is considered by careful analysis of the process itself: what are the possible influences on the subject of interest and how do they relate to each other? We mainly ensured that model correspond with literature’s observation of the process and also checked model’s d-separation properties.

The Inspection is phased process and quality of one phase influences quality of next process. Therefore, initial approximation was that all phases of Inspection are linearly connected, as shown in figure 4.1.
However while doing experiments and entering evidence of some nodes of the model; it was clear that this model was not demonstrating Inspection process's causal relation properly. For example, the effectiveness of kick off meeting affects quality of quality of checking. If the Kick off meeting is not effective, the quality of checking is also not satisfactory; and that will consequently reduce chances of having effective Inspection process. However, this is not true. In any particular Inspection, if quality of Kick off meeting is not good, then later by improving the quality of Logging or quality of Follow up, more defects can be removed from the work product. Thus, we can improve the effectiveness of Inspection process.

Model with these relations also fails to predict effectiveness of overall process when any phase is missing from the process. For example, researchers like Votta (1993) argue that Inspection logging meetings cost project time and does not help in comparison of no-
meeting process. Model specified above fails to predict Inspection effectiveness when Inspection process don’t have separate phase to log defects. As explained before, we want to develop a model, which comprehends all or at least the most important research available, and then allow simulating the process as per organisation’s needs to draw conclusions.

The other possible structure is shown in figure 4.2 by assuming that effectiveness of Inspection process depends on quality of different phases of the process. We thought that this structure can help to predict effectiveness of each phase. However, while doing experiments, we found that it fails to model the sequential phase-wise process, e.g. by using this model we cannot say that we should verify quality of work product using entry criteria, before we start checking the work product. Research shows that by following steps only, we can remove maximum defects from work product before they are carried into later stages of development.

The other disadvantage is that it is very difficult to initialise conditional probability table for 'Inspection Effectiveness' node as it has five parents: Quality of Entry criteria, Quality of Checking, Quality of Logging, Quality of Follow Up and Quality of Exit
Criteria. The probability table for child is huge and it is not possible to initialise such complex table with accurate data.

Adding few mediate nodes as shown in figure 4.3, gave us the preferred structure. First of all, this reduced the number of distributions to acquire for the child node – ‘% of defects removed on Exit (Inspection Effectiveness).’ This helps to initialise child nodes accurately. Because of small conditional probability table, this also improves the readability of the model and gives a better response time when the Bayesian Network model is run using tools like AgenaRisk.

We can also use the model pattern with most of other software quality improvement techniques, e.g. in Unit Testing also percentages of defects found affects percentages of
recorded. This model also reflected sequential dependency among Inspection process’s all phases. For example, using this process we can say that if percentages of defects found are less because of poor quality of checking, it will also reduce percentages of defects recorded (because less defects found). However, by improving quality of logging, we can record more defects, and thus can improve quality of overall process.
Using the results from past research, we have included following main influence factors in our Software Inspection effectiveness model. Following sections analyse the importance and relations between different factors, which at the end form a network of interrelations (Appendix B) and also helps to initialise the model.

4.2.1 Quality of Entry Criteria (Low, Medium, High)

Quality of Entry criteria affects quality of product on entry for Inspection. Through the Inspection process, lack of discipline and lack of respect for entry conditions waste time. Gilb & Graham (1993) explain that one of the most important entry conditions is mandating the use of upstream source documents to help inspect a product document. They also explain that it is a mistake to use uninspected upstream source documents for reference.

Management needs to take a lead on this. It is often managers who are actually responsible for overriding the entry criteria (Malotaux, 2005). For example, carrying out an Inspection is often mistakenly seen as fulfilling a quality process. Managers have been known to demand that Inspections proceed even when a team leader had determined that the entry condition concerning defects per page is violated.

4.2.2 Quality of Checking (Low, Medium, High)

This focuses on the quality of individual checking. Ideally, each checker should be assigned one or more specific roles, so that they focus on the identification of a particular type of defect, which others with a different focus might miss. This maximises the chances of as many unique issues as possible being found, which in turn makes for a better and more effective Inspection. Nodes Kick-off Meeting Effectiveness, Preparation Time, Reading Technique Type, Difficulty of Work Product and Quality of Inspection team members affects Quality of Checking node.
4.2.3 Quality of Logging (Low, Medium, High)

This focuses on quality of logging meetings. Gilb (1988) explains that the main purpose of a logging meeting is to record all potential defects identified during individual checking, as issues in the issue log. Quality of Logging depends on the Type of Meeting, the Quality of Inspection team-members, the Inspection Duration and the Quality of the Moderator.

4.2.4 Quality of Follow-up (Low, Medium, High)

The Quality of follow up also affects the Inspection process. According to Gilb and Graham (1993), Inspection is completed successfully if the defects / issues found have been dealt with by being sent elsewhere, and are under some form of configuration management. The Inspection leader should make sure that the editor has taken actions to correct all known defects, although the leader does not have to check the corrections himself.

4.2.5 Quality of Exit Criteria (Low, Medium, High)

Quality of Exit Criteria node is included in the model as it is important that the product when exited is of the quality, which is required, so that the software development process that will take place next on this product will not waste the developers' time.

The Inspection leader is responsible for checking that all Exit criteria have been met (Radice, 2002). Some examples of points covered in the criteria are: follow up must be complete, the checking rates must be within acceptable limits, and metrics must be recorded, and so on.
4.2.6 Kick-off Meeting Effectiveness (Low, Medium, High)

Kick-off meeting effectiveness mainly affects quality of checking only. However, as per literature (Radice, 2002), the Kick-off meeting is not compulsory for every Inspection process. The purpose of the meeting is to save time by disseminating information at the same time, and by clarifying the task. The Inspection leader should judge whether a Kick-off meeting is necessary or not due to the special nature of the material, or the special nature of the participants. In the meeting the leader and the team should discuss anything they need to discuss in order to be ready for an effective individual checking effort.

4.2.7 Preparation Time (0-30 min, 30-60 min, 1-4 hours, 4-8 hours, More than 8 hours)

Preparation time is the time that inspectors spend during preparation to detect defects in the inspected documents (Fagan, 1986). No significant direct relationship appears between effectiveness and preparation time. However, it is clear that Software Inspections may be inefficient because inspectors are not always sufficiently motivated to spend adequate time for pre-meeting preparations. In our model Preparation Time node affects Quality of Checking node.

Researchers like Halling & Biffl (2002) investigated whether preparation effort and size are correlated. They found a significant but weak relationship. This shows that the amount of preparation effort is not mainly driven by the size of the inspected documents. The Inspection leader must first estimate the preparation time needed based on the material to be inspected. These estimates should be validated with the Inspection team participants. At that time, the Inspection leader needs to get commitment from each participant that sufficient time is allocated for preparation.
Fagan (1986) explains that it is important to understand that inspectors may have other commitments causing the elapsed time during the preparation to appear longer than actually needed. For example, the time required for actual preparation may only be 4 hours, but one or more inspectors may not be able to complete the preparation for 3 days due to other commitments. This may not be desirable but when it cannot be changed, it will cause a lag in starting the Inspection meeting. As noted earlier, the Inspection leader should ensure that inspectors are given sufficient time for preparation. If the time is not sufficient, then this directly degrades quality of checking.

### 4.2.8 Reading Technique Type (Ad-hoc, Checklist-based, Scenario-based)

Reading technique type affects quality of checking. Reading techniques are guidelines that support the inspector while searching for defects by increasing the inspector’s reading focus. Thus, the type of technique used affects quality of checking.

Different types of reading techniques are:

1. **Ad-hoc reading**: Ad-hoc reading offers very little reading support as a software product is simply given to the inspector without any direction or guidelines on how to proceed through it and what to look for (Grady & Slack, 1994). However, Ad-hoc does not mean that inspectors do not inspect the inspected product systematically. The word ‘Ad-hoc’ only refers to the fact that no technical support is given to inspectors for the problem of how to detect defects in a software artefact. The defect detection fully depends on the skill, the knowledge, and the experience of the inspector.

2. **Checklist-based reading**: Checklists offer stronger support in the form of questions inspectors are to answer while reading the document. These questions concern to the quality aspects of the documents.
Blakely and Boles (1991) explains that checklist should be structured so that the quality attribute is clear to the inspector and the question give hints on how to assure the quality attribute, and so on.

3. Scenario-based reading: A more recent development in the area of reading techniques for individual defect detection in Software Inspection is Scenario-based reading (Shull & Rus, 2000). The essence of the Scenario-based reading idea is the use of the notion of Scenarios that provide custom guidance for inspectors on how to detect defects. The Scenario is a more detailed description for an inspector on how to perform the document review. Since each inspector may use a different Scenario, and each Scenario focuses on different defect types, it is clear that Scenario-based reading technique is the most efficient reading technique.

4.2.9 Difficulty of Work-product (Less, Medium, More)

Size, Complexity and type of item being inspected affects quality of checking. To reduce the complexity of conditional probability tables, this intermediate node is included in the model as a parent to three indicator nodes. The difficulty of the work-product being checked depends on the size, type and complexity of the work-product being inspected (figure 4.4).

![Figure 4.4: Nodes affecting Difficulty of Work Product](image-url)
4.2.10 Size of Work-product (Small, Medium, Large)

It is clear that difficulty of checking a work-product definitely depends on size of the same. As explained previously, some literatures show that exponential relationships exist between the inspected work-product size and the effort spent on preparation (Fagan, 1986). However, the relationship between them is very weak. This shows that the amount of preparation effort is not driven mainly by the size of the inspected documents. However, it definitely affects quality of checking.

4.2.11 Complexity of Work-product (Low, Medium, High)


4.2.12 Work-product type (Requirement Specification, Designs, Source-code, Test-plans, Test-specification, Documentation)

While developing the model, initially considered work-product types were only regarding programming language and node variables for that were ‘Modern / Old-style programming language’. However, it should be for general work-product type, that cannot be only code but also a document.

Work product type affects quality of checking. To check any specification is difficult than to check source code. However, Kelly et al (1992) explains that more research on the relation between Checklist and Work-product type is required. They also suggest that, an Error Checklist is work-product type dependent. For example, for source code checking, the weaker the language-type checking (Variable initialisations, constant naming, and so on), the larger will be the Checklist, e.g. C vs. Java.
4.2.13 Quality of Inspection Team Members (Poor, Fair, Good)

An important factor regarding Software Inspection is the human factor. A Software Inspection is driven by its participants, that is, the members of a project team. Hence, the success or failure of Software Inspection as a process for quality improvement and cost reduction heavily depends on human factor. If team members are unwilling to perform Inspections, all efforts will be deemed to fail.

Inspectors having influence on the Inspection process is undisputable. However, the exact impact requires further research. Their abilities, motivation and training given influence the way they conduct the process. Unfortunately, it is not easy to obtain this kind of information.

According to Pressman (1988), the quality of the team influences the time needed for defect collection and quality of the output. Larger teams tend to need more time to agree on a common position due to more discussions among members. Also, the types of persons and the team may vary the work results. If one very dominant person is more or less controlling the team, his opinion will dominate the results. This can be advantageous; if he is ‘good’ at his job, but can also lead have the adverse effect. On the other hand, if teams are comprised only of people that are known to agree on everything, the results may not be respected by the people performing the next process steps, rendering the steps taken so far useless.

Team-quality is a critical factor in meeting also. If the right people form the team, the issues raised can be solved smoothly, with good results in terms of quality of output. A team consisting of the wrong combination of people may consume considerably more effort. Inspection team is responsible for both quality of checking and quality of error-logging. Factors affecting Quality of Inspection team members are: Experience at Inspection Role; Training to Team Members; Team Size; Application Experience and Range of Team (Appendix B).
4.2.14 Experience at Inspection Role (Yes, No)

Another characteristic worth studying is the experience the inspector has with the specific modelling technique or process used during the Inspection (Wheeler, Brykczyński & Meeson, 1996). A better understanding of this type of knowledge is useful because the effects of experience with an Inspection process will help determine whether a novice or an expert is more effective.

4.2.15 Training to Team Members (Inadequate, Adequate)

An argument made by Doolan (1992) is that Inspections are useful mechanisms for skills transfer. In such a case, one should add more new and junior staff to the Inspection team to educate them. However, Votta (1993) argues that education by observation and participation is not effective, and that proper training courses are a better option. Therefore, in this case it is not obvious that a team should be comprised of properly trained inspectors.

Measurements of necessary training (Is training an ongoing or a one-time event? Have enough inspectors been trained?) reveal the health of the Inspection process. It is assumed that by training, e.g. in reading techniques or problem solving, each inspector's expertise can be improved to a certain point. It cannot be improved infinitely, of course. However, the point is that if lack in individual expertise is noted, it may be fixed by training. That is why individual training is included as an influence factor.
4.2.16 Team Size (1, 2-3, 4-6, 7-more)

There is no agreement on the optimal size for groups. The literature varies in its recommendations. Previous work in this area produced inconsistent results with recommended team sizes ranging from 2 to 12 (Fagan, 1976; Gilb and Graham, 1993; Grady & Slack, 1994; Madachy et al, 1993; Porter et al, 1995; Ebeneau, 1994; Weller, 1993). Industrial practice varies even within a single enterprise; for example, sizes between four and twelve have been used at AT&T (Fowler, 1986). Perhaps there is no single optimal team size. The teams must be kept as small as possible, but not too small. In fact the team size must be adapted to the size of the project.

Cockram (2001) explains that the optimal team size depends first on the relative costs of finding defects later in the life cycle and Inspection meeting duration. Optimal team size also depends on whether a team meeting is performed, and the duration of that meeting. As the cost of post-design defect detection activities increases, more inspectors are needed to optimise effectiveness. As the meeting duration increases, the optimal team size tends to decrease.

It is clear that the larger the Inspection team size, the more costly the Inspection will be. However, it is also the case that the more inspectors who read the given work-product, the more likely it is that more defects will be detected. Therefore, there is a trade-off between cost and the number of defects detected. It is important to identify the minimal team size that will maximise the number of defects found. Votta (1993) notes that determining the optimal team size for Inspections is an important contemporary research issue, and has called for more research on this topic.

Radice (2002) shows that size of the team also depends on the type of product and the environment in which an Inspection is performed. However, in absence of a clear answer, we recommend starting with one team that consists of three to four people: one author, one or two inspectors, and one moderator. After a few Inspections, the benefits of adding an additional inspector or an additional team member can be evaluated. The
evaluation should involve an examination whether an additional person or team helps
detect more defects, i.e., leads to an increase in Inspection effectiveness and defect
coverage. Of course, one must also address the question of whether the effort for the
extra person or team really pays off.

4.2.17 Application Experience (<=2 years, >2 years)

Cockram (2001) explains that application experience also affects the Inspection,
especially, the Inspection for coding. The primary consideration in selecting inspectors
is their ability to read and comprehend the work-product being inspected.

4.2.18 Range of Team (No, Yes)

Two important questions practitioners usually have about the range of team in the
Software Inspections are: 1. What roles are involved in an Inspection? 2. How should
people be selected for each role? Fagan (1976) suggests that Inspections should be
conducted by a number of people with defined roles: the moderator to lead the
Inspection; the designer of the program; the coder who is responsible for translating the
design into code and the tester who is responsible for testing the code. As explained
before, there is no common agreement on the size of team. Knowledge of available
range of team should help to decide team size. Therefore, this node is included in the
model.

4.2.19 Quality of Moderator (Poor, Fair, Good)

‘Quality of Moderator’ node is added in the model with the assumption that Inspection
will require a moderator; i.e., there will be more than one inspector. When there is only
one inspector, a separate moderator is not needed, as the assigned inspector will perform
all roles.
Gilb & Graham (1993) explain that moderator's primary responsibility is with the inspectors to ensure they have properly prepared for the Inspection meeting. If any inspector cannot prepare sufficiently, the moderator must select a backup inspector immediately. If a backup cannot be selected, then based on who is not prepared, the moderator must decide whether the Inspection can still be effective with a smaller team or should be rescheduled. Factors affecting Quality of Moderator are: Training to Moderator; Domain Knowledge and Communication Skills (Appendix B).

4.2.20 Training to Moderator (Inadequate, Adequate)

Fagan (1986) states that moderators need training in leadership and in creating synergy among Inspection team members during the process. However, according to Cockram (2001) about three years of experience is sufficient to understand the role of the moderator in an Inspection without the need for formal moderator training.

4.2.21 Domain Knowledge (Inadequate, Adequate)

Carver, Shull & Basili (2003) define domain knowledge as a ‘Self-ported level of familiarity with the general domain of the application.’ Domain knowledge affects the quality of Inspection, as without sufficient domain knowledge, the inspector may not understand what is being inspected and therefore cannot contribute beyond a certain level of understanding. This can be frustrating and annoying to the author of the Work-product.

4.2.22 Communication Skills (Poor, Fair, Good)

According to Cockram (2001) and Malotaux (2005), the moderator must be able to communicate, to listen actively and to encourage team members to communicate.
4.2.23 Type of Meeting (Electronic, Not Electronic)

Costs and benefits associated with type of meeting is ongoing issue. Therefore, this factor is added in the model. Any type of meetings cost, especially in two ways: first, they consume labour hours of all developers involved in which they cannot work on their respective tasks, and second, they delay the schedule. The reason for this is the time that passes by until all necessary participants find a common time and date to meet. In some cases with large teams this can be several weeks, thus delaying the Inspection process significantly.

Anderson & Teitelbaum (2001) explain the benefits of online electronic Inspection process over non-electronic Inspections. One advantage is that it allows the recording of Inspection data during the Inspection itself. The recording is visible to everyone as it happens. Another advantage is that the minor logs can be visible and transferred even before the session starts. The moderator can also get a good and immediate impression of the preparation performed. Work-product documents can also be highlighted or annotated and these can also be distributed before the session starts.

Online technology is not yet fully enabled to permit the best of Inspection meeting environments, but online technology presents an important alternative when teams are at multiple locations and widely distributed.

4.2.24 Inspection Duration (Inadequate, Adequate)

Inspection duration is another factor that affects the Inspection process. However, as explained by Kelly et al (1992), the Inspection duration is not a completely independent variable; it partially depends upon the rate at which errors are being found, which in turn depends upon the amount of preparation and density of errors present in the code. Inspection duration partially also depends on size of the work-product. If the work-product is very large than Inspection duration can be about 3 days. Therefore, we should
try to breakdown the product into smaller units so that the Inspection duration would be about 1 hour – this makes the process more effective. In our model, Inspection duration affects quality of logging meeting, as if adequate time is given then only inspectors will be able to record maximum possible number of defects.

4.3 Model Initialisation

After defining the structure of the model, we need to fill initial values in the prior probability tables. There are two main possible methods: 1. To research on individual nodes, collect data from literature and fill the probability-tables; 2. To develop questionnaires for node-connections and then distribute to experts, and finally fill the tables as per experts’ feedback.

Unfortunately it was problematic to follow first option because no reported data set contains all the factors incorporated in the proposed model. No previous research was found to be complete enough for building all probability tables for all considered factors, so following this method could represent a threat to the validate of the study.

Our solution is to elicit initial NPT values from the experts. We developed survey questionnaire (Appendix C), as we did not have enough reliable data to initialise the model. This was straight forward as we already have knowledge of possible influence-factors and also of most probable network-structure. The questionnaire also made it easy to provide initial data as it neither requires much training to use, nor requires new tools. The questionnaire was then forwarded to experienced inspection experts, who were randomly selected through world-wide web.

While building initial NPT, it was considered to build BBN including continuous variables. However, due to the lack of technical development, we faced many constraints. One constraint we faced is that continuous variables are not allowed to have a discrete child (Jensen, 2001). This means, for example, if ‘Quality of Checking’ node
is a child of ‘% of errors found’, then we cannot model whether quality of checking is Low, Medium or High as a consequence of percentages of errors found. Therefore, at the moment all our nodes are discrete nodes only.

For the evidence nodes such as ‘Quality of Exit Criteria’, which are at the bottom of the network, the initial distribution for each state of these variables is set to be flat over its range, i.e., the evidence has an equal probability for each state. For other nodes expert knowledge is used to provide prior conditional probability value. These tables were completed using AgenaRisk’s weighted mean expression feature. Using a brainstorming, from the survey we could make approximations about weighted relations among parent and child. For example, the survey could say that Quality of Moderator is more important for Quality of other team members for Quality of Logging. From this we could define the NPT for Quality of Logging to be a weighted mean expression as shown in figure 4.5.

![Figure 4.5: NPT for Quality of Logging](image)

In a similar way all remaining child node NPTs were constructed. Overall, the construction of the NPTs in a BBN model is probably the most time consuming part. However, by using expert survey (Appendix C) and AgenaRisk features (Agena, 2005), such as Expressions and Ranked nodes it was not difficult to initialise the model.
4.4 Model Verification

We need to verify that the model we have constructed is truly representative of the expert’s opinion. For that we have done sensitivity analysis of the model. The purpose of conducting sensitivity analysis is to determine that each node affects only its related parent and child nodes and no other node within the model (Agena, 2005). It is also to show that the initial belief used to initialise the model is consistent with the expert opinion provided. Here by sensitivity analysis we mean to determine the affect of each individual node of the model on its child nodes and in particular the affect on the top node of the network, which calculates the probability of Inspection effectiveness ('% of Defects removed on Exit' node), based on its parent nodes.

In the literature (Castillo et al, 1997), we have found two methods that we could use to do sensitivity analysis on a BBN, these are: 1. Entering evidence for one state in a node as 100% certain, then repeat this over all states in the node; 2. Entering a likelihood for a node, and then vary this slightly.

At the moment we have used only first method. The reason is that the prior probabilities that are being used for this model are not based on any real data; therefore the results we get from following second method will not mean much at this stage.

First method was easier to implement. Evidence was provided for each node within the network in turn, to set the node to the state, which is its best case condition, and the resulting network was calculated using AgenaRisk. The results for the parent nodes within the network were recorded and the affects of the data on the remainder of the network were observed. The model was then re-initialised prior to a new data item for the next node being entered. The experiment was repeated for each node being set to its worst case condition. Using the method, the individual contribution of each node in terms of its position and influence described by the initial belief within the network was determined systematically.
Here we explain the sensitivity analysis performed for part of the model. Figure 4.6 shows initial probability distribution graph and NPTs for 'Quality of Logging', 'Quality of Team Members' and 'Quality of Moderator' nodes.

![Figure 4.6: Model Verification - Initial state](image)

Now when the best case evidence (‘High’) was entered for ‘Quality of Team Members’ node, it changed the probability distribution of ‘Quality of Logging’ node, but not of ‘Quality of Moderator’ (figure 4.7) node.

![Figure 4.7: Best case for Quality of Inspection Team Members](image)

When the similar evidence was entered for ‘Quality of Moderator’, it did not change the ‘Quality of Team Members’ node probability table, but changed the value of ‘Quality of
Logging' node (figure 4.8). This explained that the quality of moderator and team members are independent to each other.

![Figure 4.8: Best case for Quality of Moderator](image)

However, when the worst or best case evidence was entered for ‘Quality of Logging’ node, it changed probability distribution of both ‘Quality of Moderator’ and ‘Quality of Team Members’ nodes (figure 4.9).

![Figure 4.9: Best case for Quality of Logging](image)

This means that ‘Quality of Moderator’ and ‘Quality of Team Members’ nodes are conditionally dependent on each other. The close observation of the probability distribution graphs in figure 4.7, 4.8, 4.9 also show that the ‘Quality of Moderator’ node affects the ‘Quality of Logging’ node more than the ‘Quality of Team members’ node.
does. This agrees with expert opinion that quality of moderator has more influence on the quality of logging.

Similar observations were done on complete model. The results of the sensitivity analysis show that the structure of the model is sound, e.g. the influence of nodes within the network verifies the conditional probability assignments made during initialisation. The results from the sensitivity analysis of the BBN model suggest that we managed to integrate the existing research into one single, more comprehensive model, which suggests that best results from conducting Software Inspections can be gained by: 1. Using experienced people to conduct Inspections; 2. Giving training to Inspection team and moderator; 3. Allowing adequate preparation time prior to logging meeting; 4. Dividing the work into small chunks of Inspections; 5. Managing complexity of product, and so on.

Finally, it should be noted again that the main objective of this sensitivity analysis was to ensure that model structure is truly representative of the Inspection process, and it also comprehends all or at least the most important research available. While initialising the model, we had to make few assumptions about the weighted relations among parent and child. For example, from survey we could identify that for Quality of Checking, Quality of Moderator is more important than Quality of Team Members, but the survey cannot say exactly what would be the degree of relative importance. We still need to do network calibration to revise the network potential, and then to do comparison of it with a model that only contains the initial belief for the performance testing. And then, to evaluate the model by comparing the results with another model which had been developed using an alternative modelling technique. However, we could not obtain any industry support to do these detailed experiments. The question of how accurately we can model & validate the effectiveness process depends on the reliable data really available in the future. Actually, our experience shows that it is extremely difficult to obtain reliable data of the required details, with which we can validate our conclusions. The weights in current expressions have been estimated by only some volunteer experts and thus the model is applicable within the organisation only where the experts gained
their experience. Since the model itself is not universal, one could not consider issues such as the external validity of the model. However, the experts involved were aware of the need to make it as general as possible. Therefore, the model could be used by companies, providing some company-specific customisation is considered. It might be needed to update model and questionnaire both if necessary to add some more factors that are important in a particular company.

At this stage, the significance is that at least we initialised and verified the model structure using some method; we constructed the model using both experts judgements and available literature; which gives a measurable approach that we can use for both improvement and motivation purposes. The satisfactory results from the sensitivity analysis gives confidence in the method used to utilise expert knowledge, but of course further validation using industry data would provide even greater confidence in the integrity and robustness of the model.

4.5 Influence Diagram for Cost Benefit Analysis

Above explained model can help to improve effectiveness of Inspection process, and thus quality of software development. In addition, it can also be used to make cost effective decisions for process improvement.

The following example shows how we can apply Influence Diagrams theory to the process of cost benefit analysis for Inspections improvement (figure 4.10). A decision node, ‘Training to Moderator’, represents a variable controlled by Inspection leader in order to get maximum return on investment. The Inspection leader may decide to give ‘Training to Moderator’ in order to improve the quality of logging, and thus to improve the effectiveness of Inspections.
In this example, the utility node ‘Cost’ gives information about the cost associated with ‘Training to Moderator’; and the utility node ‘Pay-off’ represents the overall payoff from increased percentages of defects Recorded.

Suppose that training might need an initial expense of £1000 and would result in an expected payoff of £2000. The Influence Diagram would highlight the decision, which maximises the expected utility. In other words, the Inspection leader would decide to spend on training only if the expected utility from ‘Training given’ exceeds the utility from ‘Training not given’ for a given value of resources available.

One of the goals of investing in SPI techniques like Inspections is to gain maximum return on investment by reducing overall development cost. For example, while assigning inspectors to the process, there should be the best balance between costs (e.g., number of inspectors) and benefits (e.g., number of errors detected).

Not considering a cost benefit analysis means that management and project team members will make their own implicit decisions and would focus on implementing most effective process only, no matter what it costs. However, if we cannot find a way to justify particular decision, we should not implement it. Other than, cost about training to Inspection team members, extra cost might occur while making decisions about training to moderator, increasing team-size, changing type of inspection meeting, and so on; and
making investment in only one factor might not give desire output, as they all together help to improve quality of Inspections. By utilising BBN & Influence diagram theory, more utility nodes about the cost of particular decision can be implemented in the model, and Inspection leader can make most cost effective decisions. Thus, BBNs can be efficiently usable indicators for the process change control, can uncover maximum defects, can motivate to follow particular decisions and save considerable project effort. This can also predict the amount of money gained or lost from creating and using a new and improved process.

4.6 Comparison with Other Models

Most of the published models e.g. Porter et al (1995) require knowledge of the total number of errors with the product after Inspection and correction. Porter's Inspection effectiveness model used the statistical process to optimise the number of variables from the initial set of variables identified by the cause and effect diagrams. However, Porter makes no attempts to describe the relationship between the variables other than that statistical relationship.

Porter's model also tends to demonstrate only part of the underlying problem. Many examples e.g. Gilb (1993), Fenton & Neil (1999) make it clear that the experience is an important attribute in the effectiveness of the process. Porter's model attempts to eliminate the affect of inspectors learning from the Inspection process. We believe that Inspections are effective because of the lessons learned, so any model of effectiveness should include the inspector's experience as part of the model. The systematic approach we have used in producing our model ensured that human factors are included in the model rather than factored out. Our model has shown importance of factors such as the experience of inspectors and moderators as explained previously.

The developed Software Inspection effectiveness model can be used to evaluate overall performance of the process or to fine tune a particular Inspection to make optimum use
of the time and Inspection resources available. We are proposing detailed Inspection
effectiveness model for analysing costs and benefits associated with Inspection process
only.

4.7 Application of Developed Model

We have developed a model of Software Inspection effectiveness, which is an
improvement over previous work. With the help of developed graphical executable
model, managers can assess the effects of changed Inspection process, which is easy to
read and also helps to focus on required attributes.

By applying our model of Software Inspection effectiveness before the Inspection takes
place, project managers can make better use of the Inspection resources available. Thus
our research contributes to the improvement of software productivity. Applying our
model using data collected during the Inspection will help in estimation of residual
errors in the work product. Decisions can then be made if further investigations are
required to identify errors close to point of introduction before these are carried into
later stages of development and test.
Chapter 5

Restructuring Agena Phase Model using Classified Defect Data

Abstract

Here we propose the model developed using probabilistic notions of Bayesian Belief Networks, which can provide support for implementing effective software process in a more mature way and can also be used to analyse return on investment for process improvements. By offering structure we facilitate the early identification of problems that threaten decision success and thus give better cost effective solution. The emphasis of the structured model is on reducing the risks associated with decisions and optimising available resources by assigning them where they give maximum benefits.

5.1 Introduction

SPI techniques like Software Inspections can be highly effective and they should be widely used in software development and maintenance (Humphrey, 1989). There is an impressive and growing list of evidence that SPI improve both quality and productivity (Basili et al, 2002; Conradi & Fuggetta, 2002; Curtis & Statz, 1996; Dyba, 2005; Goldenson & Herbsleb, 1995; Humphrey et al, 1991; Stelzer & Mellis, 1998; Solingen, 2004; and many more). However, there are also few cases where SPI techniques like Inspections have not been effective (Zamiska, 2005). This is because there were errors in the way they were conducted. Either the preparation was not adequate, too many people were involved, the wrong people attended, or too much material was covered at one time. To solve this problem, in the previous chapter we explained a causal model
(Appendix B) that helps to characterise the various aspects of Inspection process in a quantitative way and enables decision-makers to implement effective Inspection process.

Using developed model, we can increase the percentages of defects removed, and thus can reduce the risk of failure by improving the quality. It can also be used to analyse cost and benefits associated with Inspection process. However, implementing cost effective process only for Inspections is also not the complete solution. Managers are continually bombarded with publicity material claiming that the use of SPI technique will lead to X % productivity improvements, Y % cuts in maintenance costs, and so on. Most of the time, this is true; but SPI is extremely difficult, expensive, and risk intensive. Organisations still need more guidance before using any SPI techniques, which should answer some specific questions: how the investment in the improvement efforts at the end saves money; and what will be the economic impact of changing the software process? Here we propose a methodology, which can rapidly and easily help to make cost effective improvement decisions.

5.2 Integrated Methodology

Managers, developers, and customers sometimes oppose SPI activities like Software Inspections because they believe Inspections will cost too much and slow down the project. In reality, Inspections do not slow down the project, but defects do.

The important aspect that is missing from previous literature is taking into consideration the damage and loss that could be caused by undisciplined processes. One factor that contributes most to the cost of software is the effort required to correct defects detected late in the development process. Kelly, Sherif and Hops (1992) show that it is much cheaper to fix defects during the earlier stages of development. The number of defects detected after deployment directly correlates with the number of defects detected during system testing. Therefore, leaving all defect detection until the system testing stage can
only increase the number of defects deployed with the system. There is no doubt that Inspection costs. However, one of the benefits is that if we have injected too many defects, then removing some of them by Inspection is definitely cheaper than removing them with testing.

Both Inspection and Testing work within the verification and validation area. In some cases they do the same job. They both have the same goal, to raise the quality of the product. By using them, we can save both time and money and get a better product. But, they both cost as well, and how exactly one can know when to stop inspecting or testing work-product? For example, if testing of the last project stage is stopped too early, significant defects would be released to its intended users and the software manufacturer would incur the post-release cost of fixing resulting failures. If testing proceeds too long, the cost of testing and the opportunity cost could be substantial.

It is essential that test metrics and Inspection metrics both are collected and collated with each other, so that the effects of Inspection and other techniques can be assessed across the whole of the development process. If Inspection metrics are simply collected, and not compared to test data, then we will not know how effective the Inspections really are as the real payback comes in the savings in the testing operation. Therefore, for better cost-benefit analysis for SPI techniques like Software Inspections, we also need to compare the costs of Inspections to other defect removal activities such as testing. There is a need for the model which can integrate more than one quality improvement processes for better decision support, and which will help to find overall benefit by investing in any particular process.

Here because of the time and resources limitation, we could not develop detailed process effectiveness models for other than Inspection process; and we focused on using and changing existing software reliability models so that they can help with cost benefit analysis as well.
5.2.1 Existing Methodologies

As explained in chapter 1; a preliminary study of existing methodologies, standards and models revealed that support for software process improvement decisions have received limited attention. The overriding theme is that if a project is planned properly, if progress is tracked against the plan and if corrective actions are taken when work accomplished deviates from the plan; a project is likely to be successful. This is considered an incomplete and flawed view of projects because it does not take into account the inherent uncertainty associated with many of today’s software projects. Hardly any methodology (defined as a coherent set of methods, instructions, techniques, guidelines, or practices) takes into account issues like the degree of required information as input to the decision, the decision-making process itself, and implementation aspects.

To solve the problem mentioned above, for almost 10 years, Motorola Research Labs have been also developing an integrated toolset called ‘Quality Management Toolset’ (Gras, 2004; Waskiel et al, 2005). They have developed many Bayesian networks in the context of all Verification and Validation activities that match Motorola’s processes and product architecture. It does provide estimates of defects/faults left in the software at each stage of the development lifecycle. Indirectly, it does help to reduce the cost of quality, however, it does fail to predict or guide about particular decisions, which results in optimum ROI. Also that, the models they developed mainly only represent Motorola’s product/process design, and it might not be possible for some other organisations to use the toolset for their own gain. For example, the Inspection model Motorola developed, does not consider influence factors such as training to team, reading technique type and type of meeting; which can help to implement cost effective Inspections.

For software reliability predictions, the advanced phase-model (also called all_activities model) developed by Agena can also be applied at any phase of the software
development. The main objective of it is to be able to predict defects and defect rates at different periods during a software development project based information available at any stage of development and testing (Neil & Fenton, 1996; Fenton, Krause & Neil, 2001; Fenton & Neil 2004). The model combines various submodels (Specification/documentation, Development, and Testing and Rework) in such a way that these all together represent the phase of the software project at the given time. The generalised all_activities model looks at aspects of the software design process using combination of product, process and people metrics to perform an overall evaluation of particular software development. Furthermore, it can also be used as a part of a software management tool to predict product quality. Here, instead of developing any completely new tool, we decided to use and change the same Agena phase model so that it can also help with cost benefit analysis.

5.3 Restructuring Agena’s Phase Model

There are some software organisations with well-established software metrics programs that collect defect-classification data, which cover various parameters in each phase of the development cycle. These defect-classification data is simple to measure and mostly covers whole project life-cycles and thus can be used for the process improvement at any stage of the development. These classification schemes can help to avoid mistakes in the earlier phase of the life-cycles of future project; and thus can be definitely used for implementing cost effective software processes. However, Agena’s phase model does not use any defect-classification data. Grady (1992) suggested that the classified software defect data is one of the most important available management information sources for the software process improvement decisions; and also concluded that by ignoring defect data can lead to serious consequences for an organisation’s business. Classifying defects can be a difficult task and sometimes it can have ambiguous, overlapping and incomplete categories. However, the classification of defects is very
important and by examining the lessons learned by other organisations, one should implement or improve one's own defect classification and analysis efforts.

Among all the classification schemes, Boris Beizer's scheme (Beizer and Vintor; 1996) is more advantageous as it classifies defects thoroughly at given time point. Furthermore, the publicised data could also be used as the benchmark or guideline purposes for initialising the developed model. Here, we considered Boris Beizer's spreadsheet dump (Beizer, 1990) for combined defect data gathered from many different sources to initialise and validate our model. Part of the complete classification is summarised in following table.

<table>
<thead>
<tr>
<th>No of defects</th>
<th>Percentage of Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirements Incorrect</td>
<td>649</td>
</tr>
<tr>
<td>Requirements Logic</td>
<td>153</td>
</tr>
<tr>
<td>Requirements Completeness</td>
<td>224</td>
</tr>
<tr>
<td>Requirement Presentation</td>
<td>13</td>
</tr>
<tr>
<td>Requirement Changes</td>
<td>278</td>
</tr>
</tbody>
</table>

Total KLOC = 6877.26; Total Number of Defects = 16209.

**Table 5.1:** Summary of requirements specification related defects

Density of different types of defects directly affects respective process quality. Therefore, in the Agena Phase model, we replaced different indicator nodes, such as 'motivation to team-members' that are not always easy to collect and measure, with different defect-density nodes directly as indicator nodes to respective process quality nodes, e.g. Spec & doc process quality, Development process quality and Testing process quality (figure 5.1, 5.2 & 5.3).
Figure 5.1: Indicators of Specification and documentation process quality

- Density of Incorrect Requirements Defects
- Density of Requirements Logic Defects
- Density of Requirement Completeness Defects
- Density of Requirement Changes Defects

Figure 5.2: Indicators of Development process quality

- Density of Functionality Correctness Defects
- Density of Functionality Completeness Defects
- Density of Domain Defects
- Density of Data Access & Handling Defects
- Density of Integration & Implementation Defects

Figure 5.3: Indicators of Testing process quality

- Density of Test Design Defects
- Density of Test Execution Defects
- Density of Test Documentation Defects
While doing experiments with Agena’s current phase model we also found one major limitation with existing Agena phase model. In all software development stages defects are introduced, found and rework is then carried out; though often most defects are only found when the software product is almost finished, e.g. during the system and acceptance testing phase, or even during operation. Defects found during the testing phase have the disadvantage that the rework on the almost finished software product is very time consuming. It would have saved the development organisation a lot of time if these defects were found during an earlier development phase. Literature (Fagan, 1986; Gilb, 1988) suggests that early defect detection and removal improve the predictability of software projects and help project managers stay within schedule, since problems are exposed throughout the early development phases.

This also means that quality of specification process effects quality of testing process or may be quality of development process, e.g. if quality of requirement process is poor, chances of quality of testing process being high are less. However, current model fails to define this sequential flow of knowledge because the way few nodes are connected in the model and because of d-separation properties of these intermediate nodes – all main subnets does not influence each other directly (figure 5.4). Though, if someone manages to enter evidence for nodes like ‘Dummy spec & dev management quality’ and ‘Dummy testing & rework management quality’ – because of d-separation rules, process-quality of one phase can affect the other. However, it is very difficult to predict any value for such intermediate dummy nodes.
Therefore, in the Agena Phase model, we removed nodes ‘Dummy spec & dev management quality’, ‘Overall phase management quality’, and ‘Dummy testing & rework management quality’; and instead created new connections, so that ‘Spec & doc process quality’ directly affects ‘Development process quality’ node, and ‘Development process quality’ node has direct influence on ‘Testing process quality’ node (figure 5.5). Degree of these influences was kept very low initially and changed marginally every time to match with the original Residual Defects Post predictions.
5.4 Model Initialisation

After redefining the structure of the Model, we needed to fill the initial values in the new node probability tables. Here, we used available literature rather than expert-opinion to fill the probability-tables; mainly because, Beizer’s reported data set (Beizer, 1996) contains all the factors incorporated in the proposed new nodes and it was complete enough for building all probability tables for all new indicator nodes.

The results from the data-set were translated into the prior conditional probability values in the new nodes. These tables were completed using some of the AgenaRisk’s modelling features - all the defect-density nodes are chosen to be ‘Continuous Interval’, and available data-set helped to choose upper and lower bound for each node as well (figure 5.6).
There are wide range of functions and distributions available for use in expressions for the ‘Continuous Interval’ nodes (Agena, 2005). However, the Truncated Normal (TNormal) distribution could only be used because of the nature of the relationship among parent and children nodes (i.e. Parent - Rank node and Child - Continuous Interval node). From available data-set, we could also make approximations about weighted relationships among parent and child and this helped to define partial-expressions for each Node Probability Table (NPT) value. For example, figure 5.7 shows partial expression defined for ‘Low’ state of ‘Density of Incorrect Requirements Defects’ node.
Figure 5.7: Partial expression for 'Density of Incorrect Requirement Defects' node
5.5 Model Verification

Once we completed the construction of the model, we did basic sensitivity analysis of the model. By sensitivity analysis we mean to determine the effect of each individual indicator node of defect-density node on its parent node (process quality node) and in particular the effect on the top node of the network, which calculates the probability of 'Residual Defects post', based on evidence entered at different indicator nodes. For that we entered evidence for one state in a node as 100% certain, and then repeated this over all states in the node.

Evidence was provided for each new indicator node within the network in turn, to set the node to the state, which is its 'Worst case' condition, and the resulting NPT was observed for respective process-quality and 'Residual Defects post' nodes. The experiment was repeated for each node being set to its 'Best case' condition. The results of this small sensitivity analysis show that the new indicator nodes are implemented properly, e.g. the influence of these indicator nodes within the network verify the conditional probability assignments made during initialisation. Part of complete analysis is shown in table 5.2.
### Table 5.2: Effect of Defect-density Indicator node on Parent Node & 'Residual Defects Post' node

<table>
<thead>
<tr>
<th>NPT State</th>
<th>Density of Incorrect Requirements Defects</th>
<th>Spec &amp; doc process quality</th>
<th>Residual Defects Post</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Initial NPT</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Very Low</td>
<td>0.16</td>
<td>0.11</td>
<td></td>
</tr>
<tr>
<td>Low</td>
<td>0.19</td>
<td>0.22</td>
<td></td>
</tr>
<tr>
<td>Medium</td>
<td>0.23</td>
<td>0.33</td>
<td></td>
</tr>
<tr>
<td>High</td>
<td>0.14</td>
<td>0.21</td>
<td></td>
</tr>
<tr>
<td>Very High</td>
<td>0.26</td>
<td>0.14</td>
<td>Mean Value = 885.58</td>
</tr>
<tr>
<td><strong>Defect Density 'Very High'</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Very Low</td>
<td>0</td>
<td>0.41</td>
<td></td>
</tr>
<tr>
<td>Low</td>
<td>0</td>
<td>0.57</td>
<td></td>
</tr>
<tr>
<td>Medium</td>
<td>0</td>
<td>0.009</td>
<td></td>
</tr>
<tr>
<td>High</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Very High</td>
<td>1</td>
<td>0</td>
<td>Mean Value = 945.1</td>
</tr>
<tr>
<td><strong>Defect Density 'Very Low'</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Very Low</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Low</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Medium</td>
<td>0</td>
<td>0.014</td>
<td></td>
</tr>
<tr>
<td>High</td>
<td>0</td>
<td>0.45</td>
<td></td>
</tr>
<tr>
<td>Very High</td>
<td>0</td>
<td>0.53</td>
<td>Mean Value = 809.6</td>
</tr>
</tbody>
</table>

Also about removing 'Dummy spec & dev management quality', 'Overall phase management quality', and 'Dummy testing & rework management quality' nodes; part of the analysis shown in table 5.3 and 5.4, clarifies that by changing NPT value for any defect-density indicator node, it directly changes its respective Parent Node, e.g. respective Process Quality; and this change in the value of Process Quality node also changes probabilities of 'Residual Defects Post' node.
<table>
<thead>
<tr>
<th>Specification &amp; Process Quality</th>
<th>Development Quality</th>
<th>Residual Defects Post</th>
<th>Mean value</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Initial NPT</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Very Low</td>
<td>0.09</td>
<td>0.11</td>
<td></td>
</tr>
<tr>
<td>Low</td>
<td>0.21</td>
<td>0.22</td>
<td></td>
</tr>
<tr>
<td>Medium</td>
<td>0.34</td>
<td>0.33</td>
<td></td>
</tr>
<tr>
<td>High</td>
<td>0.21</td>
<td>0.22</td>
<td>950.13</td>
</tr>
<tr>
<td>Very High</td>
<td>0.13</td>
<td>0.11</td>
<td></td>
</tr>
<tr>
<td><strong>Specification &amp; Process Quality ‘Very High’</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Very Low</td>
<td>0</td>
<td>0.11</td>
<td>803.72</td>
</tr>
<tr>
<td>Low</td>
<td>0</td>
<td>0.22</td>
<td></td>
</tr>
<tr>
<td>Medium</td>
<td>0</td>
<td>0.33</td>
<td></td>
</tr>
<tr>
<td>High</td>
<td>0</td>
<td>0.22</td>
<td></td>
</tr>
<tr>
<td>Very High</td>
<td>1</td>
<td>0.11</td>
<td></td>
</tr>
<tr>
<td><strong>Specification &amp; Process Quality ‘Very Low’</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Very Low</td>
<td>1</td>
<td>0.11</td>
<td>1517.28</td>
</tr>
<tr>
<td>Low</td>
<td>0</td>
<td>0.22</td>
<td></td>
</tr>
<tr>
<td>Medium</td>
<td>0</td>
<td>0.33</td>
<td></td>
</tr>
<tr>
<td>High</td>
<td>0</td>
<td>0.22</td>
<td></td>
</tr>
<tr>
<td>Very High</td>
<td>0</td>
<td>0.11</td>
<td></td>
</tr>
</tbody>
</table>

**Table 5.3:** Effect of Specification process quality on Development process quality - with overall phase management quality node

<table>
<thead>
<tr>
<th>Specification &amp; Process Quality</th>
<th>Development Quality</th>
<th>Residual Defects Post</th>
<th>Mean value</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Initial NPT</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Very Low</td>
<td>0.09</td>
<td>0.32</td>
<td></td>
</tr>
<tr>
<td>Low</td>
<td>0.21</td>
<td>0.51</td>
<td></td>
</tr>
<tr>
<td>Medium</td>
<td>0.34</td>
<td>0.15</td>
<td></td>
</tr>
<tr>
<td>High</td>
<td>0.21</td>
<td>0.003</td>
<td>1131.8</td>
</tr>
<tr>
<td>Very High</td>
<td>0.13</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td><strong>Specification &amp; Process Quality ‘Very High’</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Very Low</td>
<td>0</td>
<td>0</td>
<td>902.65</td>
</tr>
<tr>
<td>Low</td>
<td>0</td>
<td>0.25</td>
<td></td>
</tr>
<tr>
<td>Medium</td>
<td>0</td>
<td>0.71</td>
<td></td>
</tr>
<tr>
<td>High</td>
<td>0</td>
<td>0.02</td>
<td></td>
</tr>
<tr>
<td>Very High</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td><strong>Specification &amp; Process Quality ‘Very Low’</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Very Low</td>
<td>1</td>
<td>0.78</td>
<td></td>
</tr>
<tr>
<td>Low</td>
<td>0</td>
<td>0.21</td>
<td></td>
</tr>
<tr>
<td>Medium</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>High</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Very High</td>
<td>0</td>
<td>0</td>
<td>1427.7</td>
</tr>
</tbody>
</table>

**Table 5.4:** Effect of Specification process quality on Development process quality - without overall phase management quality node
Here, the sensitivity analysis of the model suggests that after the changes we made, the model is still able to predict the defects and the defect rates at different periods during a software development project; and it successfully can use formally defined data, and thus can give more reliable predictions. The experiments agree with the available literature and suggest how quality of work-product at earlier life-cycle affects the quality at the later stage, though the mean value of 'Residual Defects Post' is still not as same as values given at original Agena all phase model. For now, the pattern of the actual results match with the expected results, but for more solid results more data or expert knowledge is required.

5.6 Using Classified Defect Data for Cost Benefit Analysis

The changes made in Agena phase model show how using collected defect classification data one can make quality predictions and get some results to improve the quality of the software process. Literature (Jones, 2006) concludes that until now, the defect injection rate in average organisations is almost twice that found in the best organisations and most of the development effort is still consumed by defect repairs, but it is still not clear how exactly defect found at one phase can affect at the later stages of the project. By applying restructured phase model before any phase of the project, project managers can make better use of collected defect-data and thus can make better use of available process resources.

However, here the primary research question is: how to do cost benefit analysis for software process improvements. We believe, applying new model using collected classified defect-data during any phase can help in estimation of residual errors and also cost of quality in the work product.
The following example shows how we can exactly apply Bayesian theory to the process of cost benefit analysis for software process improvements (figure 5.8). As shown in following abstracted model, density of different type of defects affects particular process quality (Specification, Development or Testing). Quality of each of these processes can also indirectly affect quality of the others; and depending on the process quality, percentages of total defects are removed.

Figure 5.8: Using Classified Defect Data for Cost Benefit Analysis

Now the company may decide to invest resources in giving some training to team members to improve the requirements defined, and thus to improve the overall Quality. The action of investing for training is added as a decision node ‘Training to Team’. We have given the link from Training to ‘Density of Incorrect Requirements Defects’, as we expect that the training to have an impact on the density of requirements defects. To measure the utility of the decision, we have added utility nodes ‘Cost’ and ‘Pay-off’ to the diagram, each contributing with one part of the total utility. The utility node Cost represents the information about the cost of training given, which the node Benefits represents the utility achieved at the end. To get the quantitative representation of the Influence Diagram, the assumed probability tables for each node were also constructed. Figure 5.9 shows the complete representation of the Bayesian Diagram.
**Figure 5.9**: Using Classified Defect Data for Cost Benefit Analysis – with NPTs

The example shows that Influence Diagram can highlight the decision, which maximises the expected utility. The company would decide to spend on training if the expected benefits after the training are more than the cost of the training. There are certain costs with improvement activity. However, they do not guarantee benefits. What to do if cost of training is 700 units and expected benefits are only 600 units for a given value of available resources? In these scenarios it is better to invest on something else, than to remove density of incorrect requirement defects.

In above example, we only analyse ROI associated with reducing density of incorrect requirement defects. However, extra cost might occur while making decisions about reducing other types of defects. To find and remove different types of defects at different stages have different levels of difficulties, and thus different costs also. By using Influence Diagram theory, more utility nodes about the cost of other decisions can be also implemented in the model, and decisions will benefit from more explicit consideration of the uncertainties affecting future costs and benefits.

Hence, such network can not only help to identify the risk areas in the process, but also gives guidance to achieve better quality in a cost-effective manner. By offering structure we facilitate the early identification of problems that threat decision success and thus
give better cost effective solution. The emphasis of the structured model is on reducing the risks associated with decisions and optimising available resources by assigning them where they give maximum benefits. One can have visibility into threats of later stage, and approach can be used to improve the chances of meeting the initial project commitments by reducing the level of uncertainty, and thus reducing the cost of unwanted mistakes.

5.7 Contributions

The application of the proposed methodology provides industry with a new more efficient way of improving software processes, and in concentrating on the more important attributes of the process to gain more benefits from invested efforts.

The proposed methodology for cost benefit analysis contributes to achieve cost effective software productivity and quality. The toolset recognises the uncertainty associated with process improvement techniques. Advantages of this approach are: it can consider the interactions among variables, and can highlight key variables and their possible implications; once these variables are identified, it may be possible to modify the process to gain maximum benefits.

Researchers and practitioners can profit from this research in different ways. Most importantly, the application of Bayesian Belief Networks & classified defect data can offer visibility into the ways in which resources relate to one another. It can determine the amount of money to be gained or lost by creating and using a new software process. Thus, it can provide support for decision-making: whether to use a new process, or revert to an old one, or to make more modifications to achieve desired benefits.
Chapter 6

Conclusions and Future Work

Software Process Improvement (SPI) is primarily about avoiding making mistakes: *prevention rather than cure*. SPI focuses on how well the quality objectives are met; that is to say, how effective the development process is at reducing the probability that errors are introduced (by Defect Prevention) or that any errors introduced go undetected (by Inspection or Testing).

Software Process Improvement is highly relevant to enhance productivity. It affects at all levels of the organisation. Business managers are interested because of the potential business benefits. Software project managers are interested because of the improved visibility into the software progress. End users are interested because of improved management of requirements and better chances of achieving schedule, cost and quality targets. Software engineers show interest because improving their personal software process has individual as well as professional advantages. In many contexts SPI techniques have become or are becoming an important part of the quality assurance effort for software products.

Among many available SPI techniques, this research first focuses on Software Inspections. Inspections are a process whereby software artefacts are examined by a group of inspectors to ensure that they meet defined set of quality constraints. Software Inspections have repeatedly proven to be effective in removing defects and thereby reducing the costs of the project.
Software Inspections impose a cost, but cost of having a poor quality product is even greater. However, because of associated uncertainty very few are able to quantify the short and long-term costs and benefits of implementing Software Inspections. Notions for implementation are confusing for many reasons, for example they do not explain details necessary to describe a process in depth. A review of current literature shows that Software Inspections have been successful; but benefits are still very variable in effectiveness depending greatly on the resources allocated. If we put efforts into defining the Inspection process, these efforts need to result in increased income or reduced costs, and a cost-benefit analysis can help to clarify the contribution. Many improvement efforts and their effects are so complex, that organisations require a specialised, systematic approach to know how to invest their resources to maximise their gains, meet their schedules, and minimize their risks.

To solve this issue, we use an AI technique called Bayesian Belief Network (BBN). Research proposes a probabilistic methodology that can help Industry for making cost effective decisions. This proposed indispensable methodology provides a graphical understandable approach to step-by-step process improvement by considering the factors necessary to establish a successful process. We firmly believe that by modelling SPI uncertainties, one may achieve a more realistic representation of the process, enable automated belief revision by means of Bayesian updating, and support prediction and guidance of future development activities.

As an initial aim, a BBN model for Inspections effectiveness was developed. The developed model provides a structure for improving Inspection process in a disciplined and consistent way. It includes all possible process, people and product metrics influencing Software Inspections. By different inputs (e.g., work product size and complexity) under different scenarios (e.g., available resources and personnel), the model can help to examine different alternatives, predict the effect of changes – either positive or negative - and select the one that most suits the project at hand. Through this developed model, collected data can be analysed. Changes to the process can be
proposed, and the new process can be constructed. Any changes in the data can confirm whether the expected improvements are achieved or not.

However, Inspections are only part of complete software development. Even if Inspections are very effective, if quality of some other validation and verification technique is poor; final output is not as per the expectations. Using all improvement techniques can save both time and money – but, they cost as well. Therefore, implementing cost effective process for only Inspections is not enough and we also focused on implementing the model which can integrate more than one quality improvement process for better decision support, and which can help to find overall benefit by investing in any particular process.

For this, because of time and resources limitation - we used defect classification data given by Beizer’s and restructured Agena’s phase model so that other than software reliability predictions, it can also help with the cost benefit analysis as well. The sensitivity analysis done on the model suggests that it can also improve the likelihood with which a software organisation can achieve its cost, quality, and productivity goals. The developed estimation network can be used both for cost effective quality control and process improvement.

### 6.1 Suggested Directions for Future Work

Research till date offers a path that, if followed, helps address process implementation problems for which uncertainty plays an important role, specifically with respect to human judgement and decision making. We strongly believe that software process uncertainty modelling – using Bayesian Belief Networks – will help organisations to better plan, schedule, estimate, and evaluate software process.
Due to the resource limitation, to initialise the model, we mainly used survey questionnaire (Appendix C) to initialise Inspection effectiveness model. The questionnaire makes it easy to provide initial data as it neither requires much training to use, nor does it require new tools. However, the survey cannot help estimate exactly weighed relations among parents and children.

The assumptions were also made about the weighed relations (relationships) among the parent and child and also while deciding the upper & lower bounds for different indicator node-states in all_activities phase model, even after the detailed data-set was available. This limitation could be overcome by collecting more data-set through various industries and utilising them to develop the partial expressions of the initial NPTs.

Also, here detailed experiments could not be possible for two main reasons. In the first place, it was not possible to validate complete phase-model or Inspection effectiveness model using academic laboratory settings. Secondly, it was not possible to find enough organisations and individuals who could collaborate in such independent experiments.

Therefore, as future work we need to deploy the model in the industry, run the more specified scenarios for specific projects and compare our model predictions with the observed values based on real data. From last few years, many companies are measuring their software processes. If we could gain any of their support, it would have been possible to critique, revise and improve the developed model so that the probability tables move from being individual estimates to being a statement of properties in the real world.

Furthermore, Beizer collected data mainly from US defence, aerospace and telecommunication companies; and the domain for Beizer's data-set was typically system software only. However, the industrial-sources (Jones, 2004) suggest that
different types of projects have different percentages of the defects for all the phases. According to these sources, the current model fails to predict the defect rates for the software other than the system software i.e. Military software, ERP software, etc.

To solve this, we need some more real data, other than system software domain. Once available, we will be able to refine the model and the findings will be generalised to a wider context beyond the current available research environment. The question of how accurately we can model the process is depends on the data available in the future only.
References and Bibliography


Web Site Accessed


O'Neil D (2005), "Don O'Neil Consultancy," Available at: http://members.aol.com/ONeillDon/, Last updated: 2005 [Accessed on 4 Nov 05]


# Appendices

## Appendix A: Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation/Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AI</td>
<td>Artificial Intelligence</td>
</tr>
<tr>
<td>BBN</td>
<td>Bayesian Belief Network</td>
</tr>
<tr>
<td>CASE</td>
<td>Computer Aided Software Engineering</td>
</tr>
<tr>
<td>CBA</td>
<td>Cost Benefit Analysis</td>
</tr>
<tr>
<td>CM</td>
<td>Configuration Management</td>
</tr>
<tr>
<td>CMM</td>
<td>Capability Maturity Model</td>
</tr>
<tr>
<td>CMMI</td>
<td>Capability Maturity Model Integration</td>
</tr>
<tr>
<td>CoQ</td>
<td>Cost of Quality</td>
</tr>
<tr>
<td>DAG</td>
<td>Directed Acyclic Graph</td>
</tr>
<tr>
<td>DoD</td>
<td>Department of Defense</td>
</tr>
<tr>
<td>ERP</td>
<td>Enterprise Resource Planning</td>
</tr>
<tr>
<td>GQM</td>
<td>Goal Question Metric</td>
</tr>
<tr>
<td>GUI</td>
<td>Graphical User Interface</td>
</tr>
<tr>
<td>ISO</td>
<td>International Standards Organisation</td>
</tr>
<tr>
<td>KPA</td>
<td>Key Practice Area</td>
</tr>
<tr>
<td>NPT</td>
<td>Node Probability Table</td>
</tr>
<tr>
<td>OS</td>
<td>Operating System</td>
</tr>
<tr>
<td>PIT</td>
<td>Process Improvement Team</td>
</tr>
<tr>
<td>QA</td>
<td>Quality Assurance</td>
</tr>
<tr>
<td>QMS</td>
<td>Quality Management System</td>
</tr>
<tr>
<td>RDBMS</td>
<td>Rational Database Management System</td>
</tr>
<tr>
<td>SDLC</td>
<td>Software Development Life Cycle</td>
</tr>
<tr>
<td>SEI</td>
<td>Software Engineering Institute</td>
</tr>
<tr>
<td>SI</td>
<td>Software Inspections</td>
</tr>
<tr>
<td>SPI</td>
<td>Software Process Improvements</td>
</tr>
<tr>
<td>SPM</td>
<td>Software Process Maturity</td>
</tr>
<tr>
<td>SQA</td>
<td>Software Quality Assurance</td>
</tr>
<tr>
<td>SQM</td>
<td>Software Quality Management</td>
</tr>
<tr>
<td>V&amp;V</td>
<td>Verification and Validation</td>
</tr>
</tbody>
</table>
Appendix B: Bayesian Belief Network for Software Inspection Effectiveness
Appendix C: Expert Questionnaire

Name: 
Organisation: 
Role: 
Relevant Experience: 

Mark the box that matches your view. If your opinion lies between two boxes, then mark the relative importance between the attributes within a section by marking the ranking box, 1 indicating the most important.

### INSPECTION EFFECTIVENESS

<table>
<thead>
<tr>
<th>1. Quality of Entry Criteria</th>
<th>Most important</th>
<th>Important</th>
<th>Neutral</th>
<th>Not important</th>
<th>Irrelevant</th>
<th>Ranking</th>
</tr>
</thead>
<tbody>
<tr>
<td>2. Quality of Checking</td>
<td>Most important</td>
<td>Important</td>
<td>Neutral</td>
<td>Not important</td>
<td>Irrelevant</td>
<td>Ranking</td>
</tr>
<tr>
<td>3. Quality of Logging</td>
<td>Most important</td>
<td>Important</td>
<td>Neutral</td>
<td>Not important</td>
<td>Irrelevant</td>
<td>Ranking</td>
</tr>
<tr>
<td>4. Quality of Follow up</td>
<td>Most important</td>
<td>Important</td>
<td>Neutral</td>
<td>Not important</td>
<td>Irrelevant</td>
<td>Ranking</td>
</tr>
<tr>
<td>5. Quality of Exit Criteria</td>
<td>Most important</td>
<td>Important</td>
<td>Neutral</td>
<td>Not important</td>
<td>Irrelevant</td>
<td>Ranking</td>
</tr>
</tbody>
</table>

### QUALITY OF CHECKING

| 1. Kick-off Meeting Effectiveness | Most important | Important | Neutral | Not important | Irrelevant | Ranking |
| 2. Preparation Time              | Most important | Important | Neutral | Not important | Irrelevant | Ranking |
| 3. Reading Technique Type        | Most important | Important | Neutral | Not important | Irrelevant | Ranking |
| 4. Difficulty of Work Product    | Most important | Important | Neutral | Not important | Irrelevant | Ranking |
| 5. Quality of Inspection Team Members | Most important | Important | Neutral | Not important | Irrelevant | Ranking |
### QUALITY OF LOGGING

1. **Quality of Inspection Team Members**

<table>
<thead>
<tr>
<th>Most important</th>
<th>Important</th>
<th>Neutral</th>
<th>Not important</th>
<th>Irrelevant</th>
<th>Ranking</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1/2/3/4</td>
</tr>
</tbody>
</table>

2. **Quality of Moderator**

<table>
<thead>
<tr>
<th>Most important</th>
<th>Important</th>
<th>Neutral</th>
<th>Not important</th>
<th>Irrelevant</th>
<th>Ranking</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1/2/3/4</td>
</tr>
</tbody>
</table>

3. **Type of Meeting**

<table>
<thead>
<tr>
<th>Most important</th>
<th>Important</th>
<th>Neutral</th>
<th>Not important</th>
<th>Irrelevant</th>
<th>Ranking</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1/2/3/4</td>
</tr>
</tbody>
</table>

4. **Inspection Duration**

<table>
<thead>
<tr>
<th>Most important</th>
<th>Important</th>
<th>Neutral</th>
<th>Not important</th>
<th>Irrelevant</th>
<th>Ranking</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1/2/3/4</td>
</tr>
</tbody>
</table>

### DIFFICULTY OF WORK PRODUCT

1. **Size of Work-product**

<table>
<thead>
<tr>
<th>Most important</th>
<th>Important</th>
<th>Neutral</th>
<th>Not important</th>
<th>Irrelevant</th>
<th>Ranking</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1/2/3</td>
</tr>
</tbody>
</table>

2. **Complexity of Work-product**

<table>
<thead>
<tr>
<th>Most important</th>
<th>Important</th>
<th>Neutral</th>
<th>Not important</th>
<th>Irrelevant</th>
<th>Ranking</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1/2/3</td>
</tr>
</tbody>
</table>

3. **Work-product Type**

<table>
<thead>
<tr>
<th>Most important</th>
<th>Important</th>
<th>Neutral</th>
<th>Not important</th>
<th>Irrelevant</th>
<th>Ranking</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1/2/3</td>
</tr>
</tbody>
</table>

### QUALITY OF INSPECTION TEAM-MEMBERS

1. **Experience at Inspection Role**

<table>
<thead>
<tr>
<th>Most important</th>
<th>Important</th>
<th>Neutral</th>
<th>Not important</th>
<th>Irrelevant</th>
<th>Ranking</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1/2/3/4/5</td>
</tr>
</tbody>
</table>

2. **Training to Team Members**

<table>
<thead>
<tr>
<th>Most important</th>
<th>Important</th>
<th>Neutral</th>
<th>Not important</th>
<th>Irrelevant</th>
<th>Ranking</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1/2/3/4/5</td>
</tr>
</tbody>
</table>

3. **Team Size**

<table>
<thead>
<tr>
<th>Most important</th>
<th>Important</th>
<th>Neutral</th>
<th>Not important</th>
<th>Irrelevant</th>
<th>Ranking</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1/2/3/4/5</td>
</tr>
</tbody>
</table>

4. **Application Experience**

<table>
<thead>
<tr>
<th>Most important</th>
<th>Important</th>
<th>Neutral</th>
<th>Not important</th>
<th>Irrelevant</th>
<th>Ranking</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1/2/3/4/5</td>
</tr>
</tbody>
</table>

5. **Range of Team**

<table>
<thead>
<tr>
<th>Most important</th>
<th>Important</th>
<th>Neutral</th>
<th>Not important</th>
<th>Irrelevant</th>
<th>Ranking</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1/2/3/4/5</td>
</tr>
</tbody>
</table>
### QUALITY OF MODERATOR

<table>
<thead>
<tr>
<th>Training to Moderator</th>
<th>Most important</th>
<th>Important</th>
<th>Neutral</th>
<th>Not important</th>
<th>Irrelevant</th>
<th>Ranking</th>
</tr>
</thead>
<tbody>
<tr>
<td>1/2/3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Domain Knowledge</th>
<th>Most important</th>
<th>Important</th>
<th>Neutral</th>
<th>Not important</th>
<th>Irrelevant</th>
<th>Ranking</th>
</tr>
</thead>
<tbody>
<tr>
<td>1/2/3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Communication Skills</th>
<th>Most important</th>
<th>Important</th>
<th>Neutral</th>
<th>Not important</th>
<th>Irrelevant</th>
<th>Ranking</th>
</tr>
</thead>
<tbody>
<tr>
<td>1/2/3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>