# Chapter 1
# Software Project and Quality Modelling Using Bayesian Networks

**Norman Fenton**
*Queen Mary, University of London, United Kingdom*

**Peter Hearty**
*Queen Mary, University of London, United Kingdom*

**Martin Neil**
*Queen Mary, University of London, United Kingdom*

**Łukasz Radliński**
*Queen Mary, University of London, United Kingdom, and University of Szczecin, Poland*

## ABSTRACT

*This chapter provides an introduction to the use of Bayesian Network (BN) models in Software Engineering. A short overview of the theory of BNs is included, together with an explanation of why BNs are ideally suited to dealing with the characteristics and shortcomings of typical software development environments. This theory is supplemented and illustrated using real world models that illustrate the advantages of BNs in dealing with uncertainty, causal reasoning and learning in the presence of limited data.*

## INTRODUCTION

Software project planning is notoriously unreliable. Attempts to predict the effort, cost and quality of software projects have foundered for many reasons. These include the amount of effort involved in collecting metrics, the lack of crucial data, the subjective nature of some of the variables involved and the complex interaction of the many variables which can affect a software project. In this chapter we introduce Bayesian Networks (BNs) and show how they can overcome these problems.

We cover sufficient BN theory to enable the reader to construct and use BN models using a suitable tool, such as AgenaRisk (Agena Ltd. 2008). From this readers will acquire an appreciation for the ease with which complex, yet intuitive, statistical models can be built. The statistical nature of BN models automatically enables them to deal with the

DOI: 10.4018/978-1-60566-758-4.ch001

uncertainty and risk that is inherent in all but the most trivial software projects.

Two distinctive types of model will be presented. The first group of models are primarily causal in nature. These take results from empirical software engineering, and using expert domain knowledge, construct a network of causal influences. Known evidence from a particular project is entered into these models in order to predict desired outcomes such as cost, effort or quality. Alternatively, desired outcomes can be entered and the models provide the range of inputs required to support those outcomes. In this way, the same models provide both decision support and trade off analysis.

The second group of models are primarily parameter learning models for use in iterative or agile environments. By parameter learning we mean that the model learns the uncertain values of the parameters as a project progresses and uses these to predict what might happen next. They take advantage of knowledge gained in one or more iterations of the software development process to inform predictions of later iterations. We will show how remarkably succinct such models can be and how quickly they can learn from their environment based on very little information.

## BACKGROUND

Before we can describe BN software project models, it is worthwhile examining the problems that such models are trying to address and why it is that traditional approaches have proved so difficult. Then, by introducing the basics of BN theory, we will see how BN models address these shortcomings.

### Cost and Quality Models

We can divide software process models into two broad categories: cost models and quality models. Cost models, as their name implies, aim to predict the cost of a software project. Since effort is normally one of the largest costs involved in a software project, we also take "cost models" to include effort prediction models. Similarly, since the "size" of a software project often has a direct bearing on the effort and cost involved, we also include project size models in this category. Quality models are concerned with predicting quality attributes such as mean time between failures, or defect counts.

Estimating the cost of software projects is notoriously hard. Molokken and Jorgensen (2003) performed a review of surveys of software effort estimation and found that the average cost overrun was of the order 30-40%. One of the most famous such surveys, the Standish Report (Standish Group International 1995) puts the mean cost overrun even higher, at 89%, although this report is not without its critics (Glass 2006). Software quality prediction, and in particular software defect prediction, has been no more successful. Fenton and Neil (1999) have described the reasons for this failure. We briefly reproduce these here since they apply equally to both cost and quality models.

1. Typical cost and quality models, such as COCOMO (Boehm 1981) and COQUALMO (Chulani & Boehm 1999) take one or two parameters which are fed into a simple algebraic formula and predict a fixed value for some desired cost or quality metric. Such parametric models therefore take no account of the inaccuracy in the measurement of their parameters, or the uncertainty surrounding their coefficients. They are therefore unable to attach any measure of risk to their predictions. Changes in parameters and coefficients can be simulated in an ad-hoc fashion to try to address this, but this is not widely used and does not arise as a natural component of the base model.
2. Parametric models cannot easily deal with missing or uncertain data. This is a major problem when constructing software process