Preface

Software Engineering is concerned with the planning, design, development, maintenance and documentation of software systems. It is well known that developing high quality software for real-world applications is complex. Such complexity manifests itself in the fact that software has a large number of parts that have many interactions and the involvement of many stakeholders with different and sometimes conflicting objectives. Furthermore, Software Engineering is knowledge intensive and often deals with imprecise, incomplete and ambiguous requirements on which analysis, design and implementations are based on.

Artificial intelligences (AI) techniques such as knowledge based systems, neural networks, fuzzy logic and data mining have been advocated by many researchers and developers as a way to improve many of the software development activities. As with many other disciplines, software development quality improves with the experience and knowledge of the developers, past projects and expert opinions. Software also evolves as it operates in changing and volatile environments. Hence, there is significant potential for using AI for improving all phases of the software development life cycle.

From the management point of view, during the course of a project, developers and managers need to make important decisions, plan and predict resources and time. Expert systems, neural networks, Bayesian networks and machine learning techniques have been found to be useful at this stage. Similarly, expert systems and ontologies have been proposed for organizing and eliciting user requirements. Natural language processing techniques can be used to understand requirements and research prototypes have been developed for automatic translation of requirements into Object Oriented and formal models as well as detecting ambiguities and incompleteness in requirements. Fuzzy logic can be used in modeling uncertainty due to linguistics in order to aid requirements engineering.

AI techniques have also been advocated for generating test data, devising optimal integration test orders and for fault detection. Data mining techniques have been proposed to help identify related components in the source code in order to aid software maintenance.

Thus, there is significant potential and research on utilizing AI for software development. This potential is being explored by a number of research groups but much of it is distributed in different sources such as international conferences like the World Congress on Computational Intelligence, Software Reliability Engineering, Genetic and Evolutionary Computation, and Neural Information Processing, that each has its own focus and community.

In creating this book, our aim has been to bring together some of the most advanced and innovative research on utilizing AI for software development in a single source. The book should be useful for researchers, SE practitioners and postgraduate students. Researchers and postgraduate students should find a wealth of interesting ideas and gain an appreciation of the state of the art and find useful pointers

to related studies throughout the book. Practitioners and software development institutions undoubtedly have the most to gain from improvements to the software engineering process. The results and experiences contained in the chapters should provide valuable insight into how to improve current practices and where greater investment of resources is likely to reap rewards.

BOOK ORGANIZATION

The book is organized in four section, reflecting key phases of the software engineering process: (1) Project Management and Cost Estimation, (2) Requirements Engineering and Specification, (3) Software Design and Implementation (4) Software Testing and Maintenance. In editing the book, we've been conscious that readers will want to select and read chapters independently of other parts of the book and have therefore allowed some duplication of material.

Section 1: Project Management and Cost Estimation

Good project planning involves many aspects: staff need to be assigned to tasks in a way that takes account of their experience and ability, the dependencies between tasks need to be determined, times of tasks need to be estimated in a way that meets the project completion date and the project plan will inevitably need revision as the project progresses. AI has been proposed for most phases of planning software development projects, including assessing feasibility, estimation of cost and resource requirements, risk assessment and scheduling.

Models for estimating software development costs, such as the COCOMO have existed for some time now and indeed are taught as part of most undergraduate programmes in computer science and software engineering. So why attempt to use any other methods, let alone AI? There are several reasons. First, it still remains the case that the chances of completing a software project on time and within budget remains low, suggesting that the estimates using these models may be not be reliable. Second, the conventional models don't cope well with uncertain parameters and lack the kind of flexibility that is needed for modern approaches such as agile development. The first part of the book presents four studies that use AI for cost estimation and planning in software development.

Chapter 1, by Fenton, Hearty, Neil and Radliński**,** presents work that has been widely regarded and cited as the leading research on use of Bayesian networks for predicting the cost and effort of software development. The chapter begins with an introduction to Bayesian networks and some of their properties that make them appropriate for estimation and classification. One of the key properties includes the ability to use a network by providing required outcomes as evidence and deducing the parameters that need controlling to achieve the desired outcomes. The chapter includes a description of the MODIST model for predicting cost and a model for predicting the number of defects. Versions of both of these have been tested at Phillips and have achieved high accuracy. These models are then extended to a model, called PRODUCTION, that allows further customisation to allow inclusion of context, such as local productivity rates, to be taken into account. These models are illustrated with examples of how they can be used to carry out risk assessment and trade-offs, showing the benefits of using Bayesian networks. After describing these models, which are applicable to traditional software development methodologies, the chapter describes how Bayesian networks can be used for agile methods. In particular, it shows how the Project Velocity, as used in Extreme Programming, can be estimated using Temporal

Bayesian networks. This model is further adapted to show that it is possible to produce Burnout charts as utilised by the Scrum method. The predictions from the models for Project Velocity and Burnout are both compared with data from actual projects and show remarkable accuracy.

Chapter 2, by Emilia Mendes, continues the theme of using Bayesian networks for cost estimation covered in the first chapter of the book. The focus, however, is on a more specific subset of applications, namely the effort required to develop web sites. The chapter begins with the challenges of estimating the effort required for web applications, which have their own characteristics, including the range of technologies available, the more diverse user base, range of domains, the greater variability in specialist knowledge and experience, and indeed the greater number of small and medium enterprises engaged in the industry. The chapter argues that these characteristics have led to projects that have rarely completed on time and are in need of specific tools for estimating effort required. It then proceeds to study the task of estimating the effort required in detail. First, a general framework for estimating effort, that is based on the size of a project and cost drivers, is developed. The size is characterised by factors such as expected number of pages, features and functionality and the cost drivers include factors such as number of developers and their average experience. Second, the chapter presents an extensive review of related research on estimating effort for developing web sites that includes a summary of the main features of the different approaches. The chapter then presents how to develop a web effort prediction model using Bayesian networks and presents a real case study that has already been successfully utilised in four projects of varying size and complexity.

Unlike the first two chapters, which present alternatives to models such as COCOMO, Chapter 3, by Gonsalves, Yamagishi, Kawabata and Itoh, aims to improve the COCOMO model more directly by use of swarm optimization methods. The chapter begins by providing the motivation for improvements, citing studies that show that the error rates when the COCOMO model is used in practice have been high. The cost estimation problem is formulated as a multi-objective optimization problem, where one of the objectives is to minimise the error rate and a second objective is to minimise model complexity. Minimising the model complexity as well as the error has the advantage that data collection to estimate parameters is simplified, hopefully also leading to greater accuracy. The chapter provides a brief review of existing methods for cost estimation, including regression analysis, case-based reasoning, and data mining methods. It also provides appropriate background knowledge on multi-objective optimization using genetic algorithms, COCOMO models and Pareto fronts. Swarm optimisation, which is based on the social behaviour of ants, birds and fish, is introduced and a multi-objective swarm optimization algorithm is described. An empirical evaluation of the algorithm is carried out on benchmark data sets available from the PROMISE repository and includes the well known COCOMO 81 data and NASA projects. The empirical evaluation confirms that the model is capable of making accurate estimates and also provides a Pareto front that allows managers to trade-off cost and model complexity.

Chapter 4, by Sarcia, Cantone and Basili, presents a study of how multi-layer feed forward neural networks can be a used for improving effort estimation models. The chapter begins by an introduction to parametric models and regression functions. It argues that we should consider using non-linear-in-the-parameter models instead of the linear models that have been adopted more commonly because they have closed form solutions. The potential advantages of the non-linear models include greater flexibility, parsimony and accuracy. A significant aim of the proposed approach is to reduce the number of parameters utilized. The approach aims to achieve this by utilizing a refined version of Principal Component Analysis (PCA), known as Curvilinear Component Analysis (CCA), to reduce the number of input variables. The CCA is implemented using an auto-associative multi-layer feed forward network, consisting

of coder and decoder sub-networks, and where the coder part provides the reduction or compression of input variables. The chapter concludes with an empirical evaluation based on the COCOMO 81 data set and shows that use of CCA can result in significant improvements to the accuracy of log-linear models for cost estimation.

Section 2: Requirements Engineering

Section 2 of the book is composed of four chapters that tackle different aspects of the Requirements Engineering (RE) phase. Chapter 5, by Leonid Kof, tackles the problem of incompleteness in natural language (NL) requirements, which is one of the fundamental problems in NL based requirements. This is often due to assumptions about the obviousness of some facts in the requirements and hence omitting to include them. The chapter presents a method that identifies the missing information and produces a message sequence chart (MSC) to reconstruct the missing information. Two Natural Language Processing (NLP) techniques, namely the Part-Of-Speech (POS) and parsing are used to translate NL scenarios into MSCs, where each MSC consists of an action, a sender and a receiver. POS associates a predefined POS category with each word and builds a parse tree. During the translation process, some NL deficiencies such as a missing sender, receiver or action are identified. A pre-processing phase of the system splits complex sentences into simpler ones that contain only one verb, making the identification of the sender, receiver and action easy. Some sentences are translated into messages, while others into conditions.

Chapter 6, by Martínez and Cano, shows how the use of Bayesian networks can improve requirements engineering. They state that there are two different ways of improving the requirements engineering phase. It can either be achieved by improving the RE process itself, or by improving the methods used in each activity or artifact used in the RE process. The chapter proposes a new method for dealing with the lack of certainty in individual requirements. RE is an iterative process and starts by building requirements, removing ambiguities, improving completeness and resolving conflicts and the whole process culminates in the contract. Here the problem lies in knowing when the requirements specification is good enough to stop the iterative process. So the Requirements Engineer has to take the uncertain decision about the stability of the requirements specification and decide on whether to stop the process or not. To deal with this situation, they use Bayesian networks, which are known to be good at modeling domains with uncertainty, and decide on the stability of the requirements. Their approach is composed of four steps: variables identification, structure identification, parameter elicitation, validation and testing. In the variables identification step, the domain expert and the Bayesian network builder must agree on the variables involved in the problem and their possible values or states. The purpose of the Bayesian network is to estimate certainties for unobservable or observable events. These events are called hypothesis events, and once they are detected, they are grouped into sets of mutually exclusive and exhaustive events to form hypothesis variables. The structure identification step is to set up the qualitative part of the model, with directed links connecting the variables in a directed acyclic graph. The parameter elicitation step aims to acquire the numbers necessary to estimate the conditional probabilities for the Bayesian network model. Finally, the purpose of the validation and testing step is to check the suitability of the Bayesian network model for the job it is designed to do, answering questions such as: Does the structure reflect the fundamental independence relationships of the problem? What is the level of predictive accuracy acquired? And, is the network sensitive to changes? The Bayesian network they develop forms the core of a system called Requisites, which is capable of assessing whether a requirements document needs further revision.

Chapter 7, by Muthu Ramachandran presents three research prototypes that demonstrate the potential benefits of utilizing knowledge based approaches to support agile methods. The first system, called SoBA, supports the use of a story board for agile software development. SoBA supports agile development by providing a tool that allows users to create story cards that capture functional and non-functional requirements. A novel aspect of the system is its use of a case based approach to retrieve similar past stories and suggest requirements, acceptance tests, and levels of risk. The second system, called .NET designer, provides design rationale for choosing appropriate architectural solutions. The system consists of a knowledge base of rules that represent best practice based on several years of experience and publicly available guidelines. A user of the .NET designer is guided through the architecture selection process, providing advice and recommendations based on the developed best practice knowledge base. The third system, called RAIS, explores the use of knowledge based systems to support reuse of component specifications. RAIS uses both language and domain knowledge to analyze reusability, assess reusability and suggest improvements for designing reusable software components. Together, the three systems show the kind of knowledge based software engineering tools that could be possible in the future.

Chapter 8, by Coulin, Zowghi and Sahraoui, presents the MUSTER CASE tool. MUSTER is a collaborative and situational tool that has been specifically designed and developed for requirements elicitation, and which utilizes, extends, and demonstrates a successful application of intelligent technologies for Computer Aided Software Engineering and Computer Aided Method Engineering. The overall objectives of the tool were identified as: 1) improve the process of requirements elicitation in terms of the time and effort required, and 2) directly address some of the common issues and challenges often encountered in practice, such as an incomplete understanding of needs and ill-defined system boundaries. In addition, the main functional areas required within the tool were established as 1) visualization, navigation, and administration through the elicitation process, 2) externalization, representation and organization of the elicited information, 3) process guidance, 4) cognitive support, 5) task automation, 6) interaction assistance for the participants, and 7) education of the users on requirements elicitation, primarily by osmosis. For the implementation of MUSTER, the authors have adopted a multi-agents plug-in architecture. The tool is an online browser-based application, and the sever-side components are based on the LAMP platform.

Section 3: Software Design and Implementation

Section 3 of the book consists of two chapters on software design and implementation. Chapter 9, by Liu, Khudkhudia, Wen, Sajja and Leu, develops a web-based system using an intelligent computational argumentation model for collaborative decision making in software development that facilitates selection of the most favoured development alternative. A major strength of the system is that it allows stakeholders to capture their development rationale from multiple perspectives and allows the assessment of arguments in an argumentation network using fuzzy logic based inference mechanisms.

Chapter 10, by Soria, Diaz-Pace, Bass, Bachmann, and Campo, describes two tools to support design decisions in the early stages of software development. The first tool, ArchE is an assistant for exploring architectural models, while the second tool, called SAME, is an assistant for the refinement of architectural models into object-oriented models. ArchE (**Arch**itecture **E**xpert) is a knowledge-based tool that helps the developer to explore architecture alternatives for quality-attribute scenarios while SAME (**S**oftware **A**rchitecture **M**aterialization **E**xplorer) is a case based reasoning tool for deriving object-oriented models based on previous materialization experiences. The outputs of ArchE serve as inputs for SAME. This

integrated tool approach is beneficial as it augments the designer's capabilities for navigating the design space and evaluating quality-attribute tradeoffs. Preliminary results have shown that tools like ArchE and SAME can help enforce quality-attribute requirements across the design process.

Section 4: Software Testing and Maintenance

Despite the wealth of research in the last two decades, software testing remains an area where, as cases of reported failures and numerous releases of software suggest, we cannot claim to have mastered. The final part of the book includes three chapters on the use of constraint satisfaction methods, use of ant colony optimization and temporal data mining for software testing and maintenance.

Chapter 11, by Nikolai Kosmatov, describes the use of constraint-based techniques for white and black box testing. It begins by motivating the importance of the testing phase, pointing out that it can account for as much as fifty percent of the total project costs. A background section introduces test criteria, such as structural, control flow, and data coverage criteria. After introducing the idea of model based testing, which is based on modeling a system using specification languages like VDM-SL and Z, the chapter gives a detailed example illustrating how to carry out black-box testing using constraint-based techniques. The example involves generating test cases for a uniprocessor scheduler. First, the scheduler is specified using B notation. Then, the model is parsed and each operation is transformed to pre/post conditions. The coverage criteria together with the conditions provide the definition of the constraint satisfaction problem which is solved using available tools to generate tests. The tests may not be ready to execute since the system may need to be in a particular state, hence some additional work may be necessary to complete the test oracle. Having illustrated use of constraint-based techniques for black box testing, the chapter describes how it could be used for white box testing. The central idea is to carry out a search over the conditions in a program but use the capabilities of a constraint-satisfaction solver to instantiate the variables. The chapter describes the steps involved and gives a small example to illustrate the method in detail. The chapter concludes with an excellent review of current problems in using constraint-based techniques for software testing and the future directions of research.

As well as testing goals such as achieving coverage and boundary cases, one must aim to minimize the size of the test data since carrying out the tests can be time consuming and redundant test cases add little value to the testing process. Chapter 12, by C. Peng Lam, proposes a novel method that aims to generate effective test data by utilizing ant colony optimization over UML Activity Diagrams (ADs). Ant colony optimization, which is based on the pheromone trail left by ants following a path to some food, is introduced and the problems that need addressing to utilize it for software testing are discussed. These include formulating a search problem by defining a quality measure, devising a method for updating the level of pheromone and developing state transition rules. The technique takes an AD and produces a directed graph whose vertices consist of object nodes, fork nodes, join nodes, and initial nodes of an AD and whose edges represent the activities. An ant colony optimization algorithm is developed specifically for generating test thread trees and illustrated on AD for an ATM system. The chapter includes a critical appraisal of the developed algorithm, suggesting that further trials are needed on larger and more complex ADs. The chapter also includes a section on future research directions that will be of interest to anyone pursuing research on the use of ant colony optimization for testing.

Chapter 13, by Lo, Cheng, Khoo and Liu, argues that software maintenance activities can form a significant proportion of the cost of developing software. Software can change late in the development cycle and often, pressures to deliver it to a market on time make it difficult to revise a specification properly

and generate tests systematically. Where this happens, which the chapter argues is more common than acknowledged, tools are needed that support re-engineering specifications from program behavior. One line of research that aims to re-engineer specifications is to adopt data mining methods to discover rules from execution traces. Once discovered, the rules can aid specification revision, program comprehension and test generation. Chapter 13 describes specific research on mining rules describing events that occur in the past. The chapter begins with a brief survey of existing work on specification mining, covering the Daikon tool, mining future temporal event rules, and research on extracting specifications from code. The chapter describes the format of the rules which are based on Linear Temporal Logic and develops a theory for temporal mining of traces that includes pruning of redundancy. The developed approach is tested on the JBoss Application Server and the Jeti instant messaging application using an adaptation of an existing sequential pattern miner.

The book concludes with a survey of AI methods in Software Engineering that we carried out specifically for the book. The book includes a number of chapters that describe seminal research and it is important to acknowledge that such a book cannot include all the useful research in the field. Hence, the survey aims to provide pointers to a number of other important studies and places this book in the context of the field.

We hope you enjoy reading the chapters of this book and find them as stimulating and interesting as we have done.

Farid Meziane and Sunil Vadera
University of Salford
April 2009