Preface

This book continues to provide a forum, which a recent book, *Software Evolution with UML and XML*, started, where expert insights are presented on the subject.

In that book, initial efforts were made to link together three current phenomena: software evolution, UML, and XML. In this book, focus will be on the practical side of linking them, that is, how UML and XML and their related methods/tools can assist software evolution in practice.

Considering that nowadays software starts evolving before it is delivered, an apparent feature for software evolution is that it happens *over all stages* and *over all aspects*. Therefore, all possible techniques should be explored. This book explores techniques based on UML/XML and a combination of them with other techniques (i.e., *over all techniques* from theory to tools).

Software evolution happens *at all stages*. Chapters in this book describe that software evolution issues present at stages of software architecturing, modeling/specifying, assessing, coding, validating, design recovering, program understanding, and reusing.

Software evolution happens *in all aspects*. Chapters in this book illustrate that software evolution issues are involved in Web application, embedded system, software repository, component-based development, object model, development environment, software metrics, UML use case diagram, system model, Legacy system, safety critical system, user interface, software reuse, evolution management, and variability modeling.

Software evolution needs to be facilitated *with all possible techniques*. Chapters in this book demonstrate techniques, such as formal methods, program transformation, empirical study, tool development, standardisation, visualisation, to control system changes to meet organisational and business objectives in a cost-effective way.

On the journey of the grand challenge posed by software evolution, the journey that we have to make, the contributory authors of this book have already made further advances.

Organisation of the Book

The book is organised into 15 chapters and a brief description of each chapter is as follows.

Chapter I, *Design Recovery of Web Application Transactions*, is by Scott Tilley, Damiano Distante, and Shihong Huang. Modern Web sites provide applications that are increasingly built to support the execution of business processes. In such a transaction-oriented Web site, poor transaction design may result in a system with unpredictable workflow and a lower-quality user experience. This chapter presents an example of the recovery of the "as-is" design model of a Web application transaction. The recovered design is modeled using extensions to the transaction design portion of the UML-based Ubiquitous Web Applications (UWA) framework. Recovery facilitates future evolution of the Web site.

Chapter II, *Using a Graph Transformation System to Improve the Quality Characteristics of UML-RT Specifications*, is by Lars Grunske. Architectural transformations are an appropriate technique for the development and improvement of architectural specifications. This chapter presents the concept of graph-based architecture evolution and how this concept can be applied to improve the quality characteristics of a software system, where the UML-RT used as an architectural specification language is mapped to a hypergraph-based datastructure, so that transformation operators can be specified as hypergraph transformation rules.

Chapter III, *Version Control of Software Models*, is by Marcus Alanen and Ivan Porres. Through reviewing main concepts and algorithms behind a software repository with version control capabilities for UML and other MOF-based models, this chapter discusses why source code- and XML-based repositories cannot be used to manage models and present alternative solutions that take into account specific details of MOF languages.

Chapter IV, *Support for Collaborative Component-Based Software Engineering*, is by Cornelia Boldyreff, David Nutter, Stephen Rank, Phyo Kyaw, and Janet Lavery. Collaborative system composition during design has been poorly supported by traditional CASE tools and almost exclusively focused on static composition. This chapter discusses the collaborative determination, elaboration, and evolution of design spaces that describe both static and dynamic compositions of software components from sources such as component libraries, software service directories, and reuse repositories. It also discusses the provision of cross-project global views of large software collections and historical views of individual artefacts within a collection.

Chapter V, *Migration of Persistent Object Models Using XMI*, is by Rainer Frömming and Andreas Rausch. Change is a constant reality of software development. With ever-changing customer requirements, modifications to the object model are required during software development as well as after product distribution. This chapter presents the conceptualisation and implementation of a tool for the automated migration of persistent object models. The migration is controlled by an XMI-based description of the difference between the old and the new object model.

Chapter VI, *PRAISE: A Software Development Environment to Support Software Evolution*, is by Chih-Hung Chang, William C. Chu, Chih-Wei Lu, YI-Chun Peng, and Don-

Lin Yang. This chapter first reviews current activities and studies in software standards, processes, CASE toolsets, and environments, and then proposes a process and an environment for evolution-oriented software development, known as PRocess and Agent-based Integrated Software development Environment (PRAISE). PRAISE uses an XML-based mechanism to unify various software paradigms, aiming to integrate processes, roles, toolsets, and work products to make software development more efficient.

Chapter VII, *Developing Requirements Using Use Case Modeling and the Volere Template: Establishing a Baseline for Evolution*, is by Paul Crowther. The development of a quality software product depends on a complete, consistent, and detailed requirement specification but the specification will evolve as the requirements and the environment change. This chapter provides a method of establishing the baseline in terms of the requirements of a system from which evolution metrics can be effectively applied. UML provides a series of models that can be used to develop a specification which will provide the basis of the baseline system.

Chapter VIII, *Formalizing and Analyzing UML Use Case View Using Hierarchical Predicate Transition Nets*, is by Xudong He. UML use case diagrams are used during requirements analysis to define a use case view that constitutes a system's functional model. Each use case describes a system's functionality from a user's perspective, but these use case descriptions are often informal, which are error-prone and cannot be formally analysed to detect problems in user requirements or errors introduced in system functional model. This chapter presents an approach to formally translate a use case view into a formal model in hierarchical predicate transition nets that support formal analysis.

Chapter IX, *Formal Specification of Software Model Evolution Using Contracts*, is by Claudia Pons and Gabriel Baum. During the object-oriented software development process, a variety of models of the system is built, but these models may semantically overlap. This chapter presents a classification of relationships between models along three different dimensions, proposing a formal description of them in terms of mathematical contracts.

Chapter X, *Visualising COBOL Legacy Systems with UML: An Experimental Report*, is by Steve McRobb, Richard Millham, Jianjun Pu, and Hongji Yang. This chapter presents a report of an experimental approach that uses WSL as an intermediate language for the visualisation of COBOL legacy systems in UML. Visualisation will help a software maintainer who must be able to understand the business processes being modeled by the system along with the system's functionality, structure, events, and interactions with external entities. Key UML techniques are identified that can be used for visualisation. The chapter concludes by demonstrating how this approach can be used to build a software tool that automates the visualisation task.

Chapter XI, *XML-Based Analysis of UML Models for Critical Systems Development*, is by Jan Jürjens and Pasha Shabalin. High quality development of critical systems poses serious challenges. Formal methods have not yet been used in industry as they were originally hoped. This chapter proposes to use the Unified Modeling Language (UML) as a notation together with a formally based tool-support for critical systems development. The chapter extends the UML notation with new constructs for describing criti-

cality requirements and relevant system properties, and introduces their formalisation in the context of the UML executable semantics.

Chapter XII, *Augmenting UML to Support the Design and Evolution of User Interfaces*, is by Chris Scogings and Chris Phillips. The primary focus in UML has been on support for the design and implementation of the software comprising the underlying system. Very little support is provided for the design or evolution of the user interface. This chapter first reviews UML and its support for user interface modeling, then describes Lean Cuisine+, a notation capable of modeling both dialogue structure and high-level user tasks. A case study shows that Lean Cuisine+ can be used to augment UML and provide the user interface support.

Chapter XIII, *A Reuse Definition, Assessment, and Analysis Framework for UML*, is by Donald Needham, Rodrigo Caballero, Steven Demurjian, Felix Eickhoff, and Yi Zhang. This chapter examines a formal framework for reusability assessment of development-time components and classes via metrics, refactoring guidelines, and algorithms. It argues that software engineers seeking to improve design reusability stand to benefit from tools that precisely measure the potential and actual reuse of software artefacts to achieve domain-specific reuse for an organisation's current and future products. The authors consider the reuse definition, assessment, and analysis of a UML design prior to the existence of source code, and include dependency tracking for use case and class diagrams in support of reusability analysis and refactoring for UML.

Chapter XIV, *Complexity-Based Evaluation of the Evolution of XML and UML Systems*, is by Ana Isabel Cardoso, Peter Kokol, Mitja Lenic, and Rui Gustavo Crespo. This chapter analyses current problems in the management of software evolution and argues the need to use the Chaos Theory to model software systems. Several correlation metrics are described, and the authors conclude the long-range correlation can be the most promising metrics. An industrial test case is used to illustrate that the behaviours of software evolution are represented in the Verhulst model.

Chapter XV, *Variability Expression within the Context of UML: Issues and Comparisons*, is by Patrick Tessier, Sébastien Gérard, François Terrier, and Jean-Marc Geib. Time-to-market is one severe constraint imposed on today's software engineers. This chapter presents a product line paradigm as an effective solution for managing both the variability of products and their evolutions. The product line approach calls for designing a generic and parameterised model that specifies a family of products. It is then possible to instantiate a member of that family by specialising the "parent" model or "framework," where designers explicitly model variability and commonality points among applications.