# Preface

Software-intensive systems such as telecom networks, mobile systems, handheld devices, sensor networks, and command and control systems are dependable systems which can impact our daily lives and the security of our society. Therefore, their design has to encompass many important attributes, such as systems and software development processes; business process management and economic models; service-orientation and design pattern; wireless and mobile technologies; requirement engineering; component middleware and component-based design; metamodels and metadata; traceability and systems evaluation; reverse engineering and design decision; architectural trade-offs and architecture-centric design; dependability, portability and reliability; CASE tools and distributed systems; input/output models; special purpose languages and parallel processing; deterministic and probabilistic systems; data schema and data provenance metrics; program generators and grid computing, open source revolution and disruptive technology; Web services and Semantic Web; and, functional integration, seamless integration and in-silico experiments. This book addresses these attributes.

Design for software-intensive systems requires adequate methodology and tool support in order to make use of and develop very large, complex systems. Software engineering environments help reduce the design costs of sizeable and intricate software systems, while improving the quality of the software produced. This book explores complex issues associated with software engineering environment capabilities for designing software-intensive systems.

The objective of this book is to provide relevant theoretical foundations, principles, methodologies, frameworks, and the latest research findings to professors, researchers, and graduate and undergraduate students working in the fields related to computer science, software engineering, manufacturing engineering, business administration, computer engineering, electrical engineering, industrial engineering, Web technology, and information systems. In addition, the book is expected to reveal innovative capabilities for designing software-intensive systems, important guides for managers and practitioners involved in the development of very large and complex software systems.

This book differs from others in the field because it is general enough to bring together principles, methodologies, and technologies, as well as estimation and reliability in a real-world guidebook that will help software managers, engineers, and customers immediately improve their way of designing and maintaining large, complex software systems. The scholarly value of *Designing Software-Intensive Systems: Methods and Principles* is to propose new building blocks for designing software-intensive systems in a XVI chapter book comprised of five sections. Each chapter stands alone; thus, readers can focus on topics of particular interest to them.

## SECTION I

Section I consists of three chapters dealing with process support specifications and design. Chapter I describes process artifacts required to support the development of large software projects. Chapter II presents modeling techniques for software-intensive systems. Chapter III shows how service orientation can be used to simplify the development of software-intensive systems.

### Chapter I

**"Differentiated Process Support for Large Software Projects,"** by Alf Inge Wang and Carl-Fredrik Sørensen, describes the required elements to model the software process, the required external resources, and the required process support provided by a process-centered environment. The framework presents the required process support from four perspectives: at the individual level, at the group level, at the team level, and at the project level. The chapter also discusses issues to be considered when planning and executing a software development process for large software projects with several levels of management. The objective of the research is to provide a guideline for evaluating tools supporting the development and management processes of large software projects.

### Chapter II

**"Modeling Techniques for Software-Intensive Systems,"** by Holger Giese, Stefan Henkler, Martin Hirsch, Vladimir Rubin, and Matthias Tichy, presents modeling principles which support the integration of several computing disciplines based on the software engineering standard UML. The chapter investigates a standard for systems engineering that integrates system engineering and software engineering for requirement analysis. In addition, UML extensions that (1) integrate traditional engineering modeling techniques with software engineering modeling techniques for the architectural design and detailed design, and (2) support the implementation, verification and validation phases for embedded systems are also presented. After outlining requirements and modeling techniques for software-intensive systems, the chapter looks into the most relevant modeling approaches involving business process engineering, software engineering, and control engineering before discussing the remaining integration problems. The chapter also discusses how to combine the UML extensions presented to provide a promising approach toward modeling of complex software-intensive systems.

### Chapter III

**"Service Orientation and Software-Intensive Systems,"** by Jaroslav Král and Michal Žemlička, proposes some solutions used to implement business processes in service-oriented systems. Service orientation is one known solution enabling the reuse of legacy systems and integration of third-party products. The chapter discusses subclasses of service-oriented systems. It shows how service orientation simplifies the development as it: (a) enables incremental development by using decomposition into small components; (b) supports tools for correct specification, debugging, and effective prototyping; and, (c) simplifies incremental maintenance and gradual modernization. Service orientation also provides management tools to support collaboration with business partners. Software-intensive systems are systems that depend on supporting software. The software is typically large and complex, and, as a rule, interacts with human users. The failure of the software implies failure of the system, usually causing substantial losses. The chapter shows that such software should be service-oriented. The interfaces of services should be well understood by users, and the services should be specific to applications and architectures. The authors present an important infrastructure of services as basic service-oriented design patterns for software-intensive systems.

## SECTION II

Section II consists of three chapters exploring requirements engineering and systems development life cycle for software-intensive systems. Chapter IV examines requirement analysis issues for designing and developing mobile systems. Chapter V discusses key concepts of evolution and change in model-driven software product-line architectures. Chapter VI proposes a set of core traceability requirements for designing software-intensive systems.

### Chapter IV

**"From Scenarios to Requirements in Mobile Client-Server Systems,"** by Alf Inge Wang, Carl-Fredrik Sørensen, Hien Nam Le, Heri Ramampiaro, Mads Nygård, and Reidar Conradi, examines requirements analysis issues to be considered when designing and developing mobile systems. The framework described consists of a process, the characterisation of scenarios, computation of complexity indicators, and determination of nonfunctional and functional requirements. One important goal of this framework is to identify the parts that are most complex and perhaps hardest to implement in a mobile support system. The objective of the chapter is to provide a methodology for requirement elicitations when developing systems support for mobile workers such as craftsmen, salesmen, emergency staff, police, public transportation drivers, and service staff. A system developer with basic knowledge in software engineering will learn how to develop requirements from the initial analysis process in the development of a mobile system to the description of priority, nonfunctional, and functional requirements of the end-system.

### Chapter V

**"Evolution in Model-Driven Software Product-Line Architectures,"** by Gan Deng, Douglas C. Schmidt, Aniruddha Gokhale, Jeff Gray, Yuehua Lin, and Gunther Lenz, deals with distributed real-time embedded product-line architectures, which are among the most difficult software-intensive systems to develop because such systems have limited resources and must communicate via the network to meet stringent real-time quality-of-service assurance and other performance requirements. The chapter surveys existing technologies and categorizes them along two dimensions based on their syntax and mechanisms for supporting evolution. A representative software-intensive system is used throughout the chapter as a case study to describe how to evolve product-line architectures systematically and minimize human intervention. The chapter also describes key concepts, such as model-driven engineering, product-line architectures, and model transformations that are important for supporting evolution in large-scale software intensive systems. Difficulties faced while evolving product-line architectures include handling unanticipated requirements, satisfying existing requirements, and changing metamodels. A successful process for product-line architectures evolution must, therefore, manage such changes effectively. To address these challenges, the chapter proposes a layered and compositional architecture to modularize system concerns and reduce the effort associated with domain evolution.

## Chapter VI

**"Traceability in Model-Driven Software Development,"** by Ståle Walderhaug, Erlend Stav, Ulrik Johansen and Gøran K. Olsen, focuses on lifelong management of relations between artifacts in model-driven software development. Traceability is about following the status of an artifact that is part of a system. The paper presents the background and the state-of-the-art of traceability in software development before proposing a set of core traceability requirements. The solution proposed for traceability is based on a generic metamodel and set of service specifications that use artifacts and their relationships to derive traces models. Over all, the objectives of the chapter are to: (a) show how useful is traceability in software development projects; (b) present a generic metamodel for lifelong traceability; and, (c) propose a method for incorporating traceability in the development process and tools.

## SECTION III

Section III consists of three chapters discussing architectural alternatives for software-intensive systems. Chapter VII discusses strategic selection considerations for architectural alternatives. Chapter VIII proposes a software architecture-centric approach for architecturing virtual reality systems using UML. Chapter IX proposes a survey of recent software architecture approaches.

## Chapter VII

**"Choosing Basic Architectural Alternatives,"** by Gerhard Chroust and Erwin Schoitsch, introduces basic architectural alternatives as a means to make some a-priori architectural assumptions and understand their effects when designing complex software-intensive systems. The alternatives considered in the chapter are classified according to five fundamental dimensions: time, location, granularity, control, and automation. For each dimension, several real examples are presented, discussing key properties of the choice such as elasticity and reversibility with respect to later modifications, the scope of uniformity of the system and the implication on chosen methods to accomplish the alternative. The chapter also discuses strategic selection considerations for architectural alternatives, and investigates some cross-influences for basic architectural alternatives. The analysis presented together with the specific examples and their key properties provide some insight for novices and seasoned designers and guide them during the early phases of system design.

## Chapter VIII

**"Architecting Virtual Reality Systems,"** by Rafael Capilla, Margarita Martínez, Francisco Nava, and Cristina Muñoz, describes how the design process of virtual reality systems can be improved using detailed software architectures. Virtual reality systems are complex intensive software systems that need a lot of effort and resources during the development phase. Because rigid and monolithic approaches have been often used in the past in the construction of such systems, maintenance and evolution activities become difficult tasks to carry out. Today, software architectures are used for designing more maintainable and modular systems but previous experiences in the virtual reality field didn't pay much attention to the usage of appropriate architecture descriptions. The chapter provides some guidance and potential solutions for certain design issues that can be applied to both immersive and nonimmersive virtual reality applications. To tackle certain problems affecting the construction of this kind of software intensive systems, the chapter proposes a software architecture-centric approach.

## Chapter IX

**"A Survey of Software Architecture Approaches,"** by Kendra M. L. Cooper, Lirong Dai, Renee Steiner and Rym Zalila Mili, presents established software architecture concepts followed by a survey of more recent approaches. The discussion begins with a presentation of established software architecture concepts, including architectural views, modularization and decomposition of the architectures, architectural styles, and architectural description languages. This is followed by a presentation of two well-known approaches: structured and object-oriented design. The more recent software architecture approaches included in this survey include aspect, agent, and component oriented approaches. The aspect oriented approach improves the ability to modularize capabilities, such as security, and availability in an architecture. The agent oriented approach improves the ability to model the social complexity of systems. The component oriented approach improves the ability to rapidly deploy high quality systems in order to meet new needs. In this research, each of these approaches has been proposed as a solution to a different problem. The survey is of interest to students, researchers and practitioners, in particular those who may be new to the area of software architecture.

## SECTION IV

Section IV consists of three chapters describing techniques and tools to evaluate and reduce the complexity of software-intensive systems. Chapter X proposes an approach for evaluating quality of service for enterprise distributed real-time and embedded systems. Chapter XI discusses the complexity and optimization of large systems. Chapter XII reviews key techniques and technologies to analyzing the behavior of software systems.

## Chapter X

**"Dynamic Analysis and Profiling of Multithreaded Systems,"** by Daniel G. Waddington, Nilabja Roy and Douglas C. Schmidt, reviews key techniques and technologies to analyzing the behavior of software systems via dynamic analysis: compiler based instrumentation, operating system and middleware profiling, virtual machine profiling, and hardware-based profiling. Dynamic analysis is weaker and less capable when the behavioral characteristics of concern depend on a system-wide analysis such as the global thread state at a given point in time. The alternative to dynamic analysis is static analysis. The benefits of static analysis are its ability to (1) perform analysis without running the system and (2) allow inspection of all theoretically possible conditions. Although static analysis shows some promise for selected applications it cannot predict and present a complete picture of behavior for larger-scale systems. The chapter highlights the advantages and disadvantages of each approach with respect to measuring the performance of multithreaded systems, and demonstrates how these approaches can be combined and applied in practice.

## Chapter XI

**"Evaluating Quality of Service for Enterprise Distributed Systems,"** by James H. Hill, Douglas C. Schmidt, and John M. Slaby, finds out that existing techniques for evaluating the performance of component architectures are relatively complex and at the middleware infrastructure level only. What is needed, therefore, are next-generation modeling tools that provide the same analysis techniques, but shield the developer from the complexity of existing tools. The chapter describes existing techniques and tools used to evaluate the quality of service of some enterprise systems. To motivate the structure and functionality of next-generation tools, the chapter presents a service-oriented architecture case study for naval shipboard computing systems and the challenges encountered while developing and evaluating it. The main goals of this case study are to simplify the process of determining which deployment and configuration strategies will meet deadlines, and to reduce the time for integrating and testing the actual system components after they are completed. The chapter combines system execution modeling tools with model-driven engineering technologies to address the development and integration challenges of service-oriented architecture-based enterprise software-intensive systems. The research showed how an existing modeling tool could be applied to the case study presented to address integration challenges during early stages of development.

### Chapter XII

**"Reducing the Complexity of Modeling Large Software Systems,"** by Jules White, Douglas C. Schmidt, Andrey Nechypurenko and Egon Wuchner, deals with reducing the complexity of modeling and optimizing large systems through the integration of constraint solvers with graphical modeling tools. For large-scale systems, traditional modeling approaches allow developers to raise the level of abstraction used for solution specification and illuminate design flaws earlier in the development cycle. Such systems, however, have exponential design constraints, extremely large model sizes, or other complexities that make it hard to handcraft a model of a solution. For these types of challenging domains, automated design guidance based on the design constraints is needed. The chapter illustrates the specific challenges of using model-driven development tools for these types of complex domains. The chapter also presents techniques based on integrating constraint solvers into modeling environments that can be used to address these challenges and also illuminates the complexities of leveraging constraint solvers and describes effective techniques and tools to manage the complexity they introduce.

## SECTION V

Section V consists of four chapters dealing with best practices and integrations in software-intensive systems. Chapter XIII proposes a domain-specific language for describing grid applications. Chapter XIV discusses market forces for understanding the open source revolution. Chapter XV presents system integration concepts for distributed enterprise systems. Chapter XV proposes mechanisms for evaluating data provenance in data intensive domains.

### Chapter XIII

**"A Domain-Specific Language for Describing Grid Applications,"** by Enis Afgan, Purushotham Bangalore, and Jeff Gray, introduces grid computing concepts and provides a taxonomy of grid users. The chapter builds on grid technologies and provides examples of grid application development and deployment through a sample scenario outlining some of the difficulties with the current grid technologies. By adopting the grid, the requirements related to application deployment have risen and require new expertise, often not readily and easily available due to the involved complexities. This implies the need for a grid specialist who possesses deep understanding of the technology. The drawback to this solution is that additional communication is necessary at the development level, which prolongs the application development in addition to possibilities to introduce errors due to miscommunication or misunderstanding. These issues are addressed in the chapter with an introduction of a new language. The construction of this language is eased by a support tool based on metamodeling technologies.

## Chapter XIV

**"A Framework for Understanding the Open Source Revolution,"** by Jeff Elpern and Sergiu Dascalu, presents open source development as a fundamentally new paradigm driven by economics and facilitated by new processes. Development theory, methodologies, processes and techniques have mostly evolved from the environment of proprietary, large-scale and large-risk software systems. The software design principles function within a hierarchical decision-making framework. Development of banking, enterprise resource and complex weapons systems all fit this paradigm. However, as this chapter describes, another paradigm for the design and implementation of software-intensive systems has emerged. The open source management structure is "flat." The development team is dispersed and works for many different organizations. The testing processes put most of the work in the user community. Judged by the methodologies of large-scale, proprietary development, open source projects look like chaos. However, the real-world results have been spectacular. The chapter proposes a framework for understanding the open source revolution by identifying a number of market forces driving the revolution and placing these forces within historical perspectives.

## Chapter XV

**"Quality Metrics for Evaluating Data Provenance,"** by Syed Ahsan and Abad Shah, determines and measures the data quality and its needs in data intensive domains. Data provenance has been proposed as one of the many quality metrics to determine the quality of data. So far, data provenance itself has not been investigated in detail. The chapter gives a literature survey of the field and discusses issues and problems of data provenance. The mechanisms of collecting metadata to establish data provenance, the number of parameters which affect the quality of provenance, and important characteristics which must be part of any data provenance are discussed in the paper. The chapter also identifies different applications of data provenance in various domains, and proposes a set of parameters and related metrics to measure data provenance. These proposed metrics can assist software researchers and practitioners to better understand and support data provenance in various domains such as software-intensive systems.

## Chapter XVI

**"System Integration Using Model-Driven Engineering,"** by Krishnakumar Balasubramanian, Douglas C. Schmidt, Zoltán Molnár, and Ákos Lédeczi, contributes to functional integration in distributed enterprise systems. With the emergence of commercial-off-the-shelf components, software integrators are increasingly faced with the task of integrating heterogeneous enterprise distributed systems built using different technologies. Although there are well-documented patterns and techniques for system integration using various middleware technologies, system integration is still largely a tedious and error-prone manual process. To improve this process, the chapter proposes that component developers and system integrators must understand key properties of the systems they are integrating, as well as the integration technologies they are applying. The chapter describes the challenges associated with functionally integrating software for these types of systems and presents how composition of domain-specific modeling languages can be used to simplify functional integration. Using a case study, the research demonstrates how metamodel composition can solve functional integration by reverse engineering an existing system and exposing it as Web services to Web clients who use these services. Metamodeling provides significant benefits with respect to automation and reusability compared to conventional integration processes and methods.

**Pierre F. Tiako** *is the director of the Center for Information Technology Research at Langston University (USA) and an assistant professor of computer science and information systems. He worked as a visiting professor at Oklahoma State University (OSU) before the current position. Prior to OSU, he taught computer science courses and did research at Universities of Nancy and Rennes (France), and also worked as an expert engineer at INRIA, the French National Institute for Research in Information Technology. Dr. Tiako has authored more than 50 journal and conference technical papers and coedited four proceedings volumes, resulting from services as program chair for several international conferences and workshops. He holds a PhD in software and information systems engineering from National Polytechnic Institute of Lorraine (France). Dr. Tiako is a senior member of IEEE and past chairman for IEEE Oklahoma City Computer Society.*