Preface

The book begins with “A Link-Based Ranking Algorithm for Semantic Web Resources: A Class-Oriented Approach Independent of Link Direction” by Hyunjung Park et al. The information space of the Semantic Web has different characteristics from that of the World Wide Web (WWW). One main difference is that in the Semantic Web, the direction of Resource Description Framework (RDF) links does not have the same meaning as the direction of hyperlinks in the WWW, because the link direction is determined not by a voting process but by a specific schema in the Semantic Web. Considering this fundamental difference, the authors propose a method for ranking Semantic Web resources independent of link directions and show the convergence of the algorithm and experimental results. This method focuses on the classes rather than the properties. The property weights are assigned depending on the relative significance of the property to the resource importance of each class. It solves some problems reported in prior studies, including the Tightly Knit Community (TKC) effect, as well as having higher accuracy and validity compared to existing methods.

Chapter 2, “A Study of Open Source Software Development from Control Perspective” by Bo Xu et al., describes how open source software (OSS) has achieved great success and exerted significant impact on the software industry. OSS development takes online community as its organizational form, and developers voluntarily work for the project. In the project execution process, control aligns individual behaviors toward the organizational goals via the Internet and becomes critical to the success of OSS projects. This chapter investigates the control modes in OSS project communities, and their effects on project performance. Based on a web survey and archival data from OSS projects, it is revealed that three types of control modes, that is, outcome, clanship, and self-control, are effective in an OSS project community. The study contributes to a better understanding of OSS project organizations and processes, and provides advice for OSS development.

Next, Jesús Pardillo et al. investigate the customization of data warehouses. To customize a data warehouse, many organizations develop concrete data marts focused on a particular department or business process. However, the integrated development of these data marts is an open problem for many organizations due to the technical and organizational challenges involved during the design of these repositories as a complete solution. In this chapter, “An MDA Approach and QVT Transformations for the Integrated Development of Goal-Oriented Data Warehouses and Data Marts,” the authors present a design approach that employs user requirements to build both corporate data warehouses and data marts in an integrated manner. The approach links information requirements to specific data marts elicited by using goal-oriented requirement engineering, which are automatically translated into the implementation of corresponding data repositories by means of model-driven engineering techniques. The authors provide two UML profiles that integrate the design of both data warehouses and data marts.
and a set of QVT transformations with which to automate this process. The advantage of this approach is that user requirements are captured from the early development stages of a data-warehousing project to automatically translate them into the entire data-warehousing platform, considering the different data marts. Finally, the authors provide screenshots of the CASE tools that support the approach, and a case study to show its benefits.

In the next chapter, “Understanding Business Domain Models: The Effect of Recognizing Resource-Event-Agent Conceptual Modeling Structures,” Geert Poels investigates the effect on understanding of using business domain models that are constructed with Resource-Event-Agent (REA) modeling patterns. First, the author analyzes REA modeling structures to identify the enabling factors and the mechanisms by means of which users recognize these structures in a conceptual model and description of an information retrieval and interpretation task. Based on this understanding, the author hypothesizes positive effects on model understanding for situations where REA patterns can be recognized in both task and model. An experiment is then conducted to demonstrate a better understanding of models with REA patterns compared to information equivalent models without REA patterns. The results of this experiment indicate that REA patterns can be recognized with minimal prior patterns training and that the use of REA patterns leads to models that are easier to understand for novice model users.

In Chapter 5, Marco Crasso et al. conduct a review of research on “A Survey of Approaches to Web Service Discovery in Service-Oriented Architectures.” Discovering services acquires importance as Service-Oriented Computing (SOC) becomes an adopted paradigm. SOC’s most popular materializations, namely Web Services technologies, have different challenges related to service discovery and, in turn, many approaches have been proposed. As these approaches are different, one solution may be better than another according to certain requirements. In consequence, choosing a service discovery system is a hard task. To alleviate this task, this chapter proposes eight criteria, based on the requirements for discovering services within common service-oriented environments, allowing the characterization of discovery systems. These criteria cover functional and non-functional aspects of approaches to service discovery. The results of the characterization of 22 contemporary approaches and potential research directions for the area are also shown.

K. Vidyasankar and Gottfried Vossen describe “Multi-Level Modeling of Web Service Compositions with Transactional Properties.” Web services have become popular as a vehicle for the design, integration, composition, reuse, and deployment of distributed and heterogeneous software. However, although industry standards for the description, composition, and orchestration of Web services have been under development, their conceptual underpinnings are not fully understood. Conceptual models for service specification are rare, as are investigations based on them. This chapter presents and studies a multi-level service composition model that perceives service specification as going through several levels of abstraction. It starts from transactional operations at the lowest level and abstracts into activities at higher levels that are close to the service provider or end user. The authors treat service composition from a specification and execution point of view, where the former is about composition logic and the latter about transactional guarantees. Consequently, the model allows for the specification of a number of transactional properties, such as atomicity and guaranteed termination, at all levels. Different ways of achieving the composition properties and implications of the model are presented. The authors also discuss how the model subsumes practical proposals like the OASIS Business Transaction Protocol, Sun’s WS-TXM, and execution aspects of the BPEL4WS standard.

Next, “Service Composition and Interaction in a SOC Middleware Supporting Separation of Concerns with Flows and Views” by Dickson K. W. Chiu et al. explores Service-Oriented Computing (SOC),
which has recently gained attention both within industry and academia; however, its characteristics cannot be easily solved using existing distributed computing technologies. Composition and interaction issues have been the central concerns, because SOC applications are composed of heterogeneous and distributed processes. To tackle the complexity of inter-organizational service integration, the authors propose a methodology to decompose complex process requirements into different types of flows, such as control, data, exception, and security. The subset of each type of flow necessary for the interactions with each partner can be determined in each service. These subsets collectively constitute a process view, based on which interactions can be systematically designed and managed for system integration through service composition. The authors illustrate how the proposed SOC middleware, named FlowEngine, implements and manages these flows with contemporary Web services technologies. An experimental case study in an e-governmental environment further demonstrates how the methodology can facilitate the design of complex inter-organizational processes.

Web services are defined independently of any execution context. Due to their inherent autonomy and heterogeneity, it is difficult to examine the behaviour of composite services, especially in case of failures. Chapter 8, “Ensuring Customised Transactional Reliability of Composite Services” by Sami Bhiri et al. is interested in ensuring composite services reliability. Reliable composition is defined as a composition where all instance executions are correct from a transactional and business point of view. In this chapter, the authors propose a transactional approach for ensuring reliable Web service compositions. The approach integrates the expressivity power of workflow models and the reliability of Advanced Transactional Models (ATM). This method offers flexibility for designers to specify their requirements in terms of control structure, using workflow patterns, and execution correctness. Contrary to ATM, the authors start from the designers’ specifications to define the appropriate transactional mechanisms that ensure correct executions according to their requirements.

In the following chapter, Hiroshi Wada et al. write about “Leveraging Early Aspects in End-to-End Model Driven Development for Non-Functional Properties in Service Oriented Architecture.” In Service Oriented Architecture (SOA), each application is designed with a set of reusable services and a business process. To retain the reusability of services, non-functional properties of applications must be separated from their functional properties. This chapter investigates a model-driven development framework that separates non-functional properties from functional properties and manages them. This framework proposes two components: (1) a programming language, called BALLAD, for a new per-process strategy to specify non-functional properties for business processes, and (2) a graphical modeling method, called FM-SNFPs, to define a series of constraints among non-functional properties. BALLAD leverages aspects in aspect oriented programming/modeling. Each aspect is used to specify a set of non-functional properties that crosscut multiple services in a business process. FM-SNFPs leverage the notion of feature modeling to define constraints among non-functional properties like dependency and mutual exclusion constraints. BALLAD and FM-SNFPs free application developers from manually specifying, maintaining and validating non-functional properties and constraints for services one by one, reducing the burdens/costs in development and maintenance of service-oriented applications. This chapter describes the design details of BALLAD and FM-SNFPs, and demonstrates how they are used in developing service-oriented applications. BALLAD significantly reduces the costs to implement and maintain non-functional properties in service-oriented applications.

Chapter 10, “Complementing Business Process Verification by Validity Analysis: A Theoretical and Empirical Evaluation” by Pnina Soffer and Maya Kaner, investigates the need for complementing automated verification of business process models with a validity analysis performed by human analysts.
As business processes become increasingly automated through process aware information systems, the quality of process design becomes crucial. Although verification of process models has gained much attention, their validation, relating to the reachability of the process goal, has hardly been addressed. The chapter investigates the need for model validation both theoretically and empirically. The authors present a theoretical analysis, showing that process model verification and validation are complementary in nature, and an empirical evaluation of the effectiveness of validity criteria in validating a process model. The theoretical analysis, which relates to different aspects of process model quality, shows that process model verification and validation are complementary in nature. The empirical findings corroborate the effectiveness of validity criteria and indicate that a systematic criteria-supported validity analysis improves the identification of validity problems in process models.

In “Data Management and Data Administration: Assessing 25 Years of Practice,” Peter Aiken et al. take a retrospective view of Data Management (DM), which has existed in conjunction with software development and the management of the full set of information technology (IT)-related components. However, it has been more than two decades since research into DM as it is practiced has been published. In this chapter, the authors compare aspects of DM across a quarter-century timeline, obtaining data using comparable sets of subject matter experts. Using this information to observe the profession’s evolution, the authors have updated the understanding of DM as it is practiced, giving additional insight into DM, including its current responsibilities, reporting structures, and perceptions of success, among other factors. The analysis indicates that successfully investing in DM presents current, real challenges to IT and organizations. Although DM is evolving away from purely operational responsibilities toward higher-level responsibilities, perceptions of success have fallen. This chapter details the quarter-century comparison of DM practices, analyses them, and draws conclusions.

The quality of conceptual models directly affects the quality of the understanding of the application domain and the quality of the final software products that are ultimately based on them. “A Systematic Literature Review on the Quality of UML Models,” by Marcela Genero et al., describes a systematic literature review (SLR) of peer-reviewed conference and journal articles published from 1997 through 2009 on the quality of conceptual models written in UML, undertaken to understand the state-of-the-art, and then identify any gaps in current research. Six digital libraries were searched, and 266 papers dealing specifically with the quality of UML models were identified and classified into five dimensions: type of model quality, type of evidence, type of research result, type of diagram, and research goal. The results indicate that most research focuses on semantic quality, with relatively little on semantic completeness; as such, this chapter examines new modeling methods vs. quality frameworks and metrics, as well as quality assurance vs. understanding quality issues. The results also indicate that more empirical research is needed to develop a theoretical understanding of conceptual model quality. The classification scheme developed in this chapter can serve as a guide for both researchers and practitioners.

In “Semi-Automatic Composition of Situational Methods,” Anat Aharoni and Iris Reinhartz-Berger define situational methods as approaches to the development of software systems that are designed and constructed to fit particular circumstances that often refer to project characteristics. One common way to create situational methods is to reuse method components, which are the building blocks of development methods. For this purpose, method components must be stored in a method base, and then retrieved and composed specifically for the situation in hand. Most approaches in the field of situational method engineering require the expertise of method engineers to support the retrieval and composition of method components. Furthermore, this is usually done in an ad-hoc manner and for pre-defined situations. In this chapter, the authors propose an approach, supported by a tool that creates situational methods semi-
automatically. This approach refers to structural and behavioral considerations and a wide variety of characteristics when comparing method components and composing them into situational methods. The resultant situational methods are stored in the method base for future usage and composition. Based on an experimental study of the approach, the authors show that it provides correct and suitable draft situational methods, which human evaluators have assessed as relevant for the given situations.

The next chapter, “DocBase: Design, Implementation, and Evaluation of a Document Database for XML” by Arijit Sengupta and Ramesh Venkataraman introduces a complete storage and retrieval architecture for a database environment for XML documents. DocBase, a prototype system based on this architecture, uses a flexible storage and indexing technique to allow highly expressive queries without the necessity of mapping documents to other database formats. DocBase is an integration of several techniques that include: (i) a formal model called Heterogeneous Nested Relations (HNR), (ii) a conceptual model XER (Extensible Entity Relationship), (ii) formal query languages (Document Algebra and Calculus), (iii) a practical query language (Document SQL or DSQL), (iv) a visual query formulation method with QBT (Query By Templates), and (v) the DocBase query processing architecture. This chapter focuses on the overall architecture of DocBase including implementation details, describes the details of the query-processing framework, and presents results from various performance tests. The chapter summarizes experimental and usability analyses to demonstrate its feasibility as a general architecture for native as well as embedded document manipulation methods.

Keng Siau et al. develop “A Meta-Analysis Comparing Relational and Semantic Models” in the next chapter. Data modeling is the *sine quo non* of systems development and one of the most widely researched topics in the database literature. In the past three decades, semantic data modeling has emerged as an alternative to traditional relational modeling. The majority of the research in data modeling suggests that the use of semantic data models leads to better performance; however, the findings are not conclusive and are sometimes inconsistent. The discrepancies that exist in the data modeling literature and the relatively low statistical power in the studies make meta-analysis a viable choice in analyzing and integrating the findings of these studies.

The final chapter, “Extending Agile Principles to Larger, Dynamic Software Projects: A Theoretical Assessment” by Dinesh Batra et al. evaluates the feasibility of extending agile principles to larger, dynamic, and possibly distributed software development projects by uncovering the theoretical basis for agile values and principles for achieving agility. The extant literature focuses mainly on one theory—complex adaptive systems—to support agile methods, although recent research indicates that the control theory and the adaptive structuration theory are also applicable. This chapter proposes that at least three other theories exist that are highly relevant: transaction cost economics, social exchange theory, and expectancy theory. By employing these theories, a rigorous analysis of the Agile Manifesto is conducted. Certain agile values and principles find theoretical support and can be applied to enhance agility dynamic projects regardless of size; some agile principles find no theoretical support while others find limited support. Based on the analysis and the ensuing discussion, the authors propose a framework with five dimensions of agility: process, design, people, outcomes, and adaptation.

*Keng Siau  
University of Nebraska-Lincoln, USA*