Preface

While editing this book in the year 2011, we realized that the concept of software engineering was forty-three years old. The term "software engineering" was coined at the first NATO Software Engineering Conference in Germany in 1968 amid widespread consensus that problems were emerging with software development and maintenance. These problems were later widely discussed, and the term "software crisis" emerged to describe the software industry's inability to provide customers with high-quality products within schedule and under budget. Hardware costs were dropping, while software costs were rapidly rising. Major computer system projects arrived years late, and the resulting software was unreliable, hard to maintain, and performed poorly. Today, the situation is not much changed. Software development projects are notorious for being completed far over budget and behind schedule. A survey conducted by the Standish Group in the United States (US) in 2008 examined data from several thousand information technology (IT) projects and revealed a software project success rate of only 32%. Twenty-four percent of projects failed, whereas the remaining 44% had cost overruns, time overruns, and impaired functionalities.

The processes of large-scale software development can be vast and complex, involving many software engineers, programmers, and designers. As a result, they are often hard to define, difficult to understand, and even harder to establish. For these reasons, software process improvement (SPI) models were developed. The use of SPI is based on the premise that mature and capable processes generate quality software products. Over the decades, several models for improving software quality through management of the software process have become significant in the software industry. The importance of the software process concept is indicated by the widespread popularity of the software process and software process improvement approaches in software development organizations around the world. To this end, various efforts, initiatives, models, methodologies, and standards have been developed in the last few years, and exhaustive lists of models are available. As members of the software engineering community, we realized that we must continue to promote the education of software engineers to develop and expand knowledge in this area. We believe that the implementation and proper management of software process improvement will lead to the production of high-quality software products.

Software process improvement and management is now such an enormous discipline that is impossible to cover the whole area of knowledge in one book. Therefore, this book gathers the latest advances in various topics in software process improvement and reports on ways that organizations and companies can gain a competitive advantage by applying various emergent methodologies and techniques to real-world scenarios. We focus on emerging key topics, including software process lines, requirement engineering process improvement, process standards and improvement for very small enterprises, software process improvement, software process assessment, process tools and automated support, and software process

improvement benefits. With 11 chapters written by 29 international experts in the area of software process improvement and management, this book is intended to serve as a standard reference.

The software industry has come a long way since its early days as a cottage industry. In its short history, it has passed through a number of profound changes and waves of changes, moving it firmly toward its status as a discipline, mirroring what happened to manufacturing during the Industrial Revolution. The **first chapter** of this book positions the concept of Software Product Lines as a possible foundation for software industrialization, and the authors introduce the concept of Software Process Lines as a complementary foundation for software industrialization. The authors believe that an era of software industrialization is coming in spite of the difficulties facing software development efforts. In this chapter, the authors discuss the justifications and benefits of Software Process Lines. They describe the steps for implementing Software Process Lines and explain how Software Process Lines can enable and facilitate the establishment of a continuous Software Process Improvement environment. Because the idea of Software Process Lines is very new, the authors suggest that further research is needed to move the concept forward.

Chapter 2 focuses on Requirement Engineering (RE) process improvement models. It presents a review of selected requirement engineering-specific and generic process improvement models available in the public domain. The review aims to provide preliminary information that might be needed by organizations when selecting these models. The chapter begins with an analysis of the way RE maturity is addressed in the Capability Maturity Model Integration (CMMI) for Development. It then describes both the principal characteristics of and the assessment and improvement framework applied in four RE-specific process assessment and improvement models: the Requirements Engineering Good Practice Guide (REGPG), the Requirements Engineering Process Maturity (REPM), the Requirements Capability Maturity Model (R-CMM), and the Market-Driven Requirements Engineering Process Model (MDREPM). It also examines the utility of, and lessons learned from, these models.

Most software engineering centers, such as the Software Engineering Institute (SEI), dedicate a large portion of their resources to large organizations. Although there seems to be an awareness of the needs of Very Small Enterprises (VSEs), published software engineering practices are still, for the most part, unusable by organizations with less than 25 people. Only a few centers around the world are focusing their Software Process Improvement (SPI) activities on small enterprises and VSEs. Many international standards and models, such as ISO/IEC 12207 or CMMI, have been developed to capture proven engineering practices. However, these documents were not designed for VSEs and are often difficult to apply in such settings. **Chapter 3** presents a description of the development of international process improvement standards (ISs) targeting VSEs developing software as a standalone product or software as a component of a system. The documents used by the ISO/IEC JTC1/SC7 Working Group 24 (WG24) and the approaches that led to the development, and balloting of the ISs and Technical Reports (TRs) for VSEs are also presented. The chapter also focuses on the ISO/IEC 29110 Standard, the development of means of helping VSEs improve their processes and the description of a few pilot projects conducted to implement the processes of the 29110 standard.

Chapter 4 aims to improve quality in software engineering projects by introducing Verification and Validation (V&V): best practices in terms of process, artifacts, and quality performance indicators. It presents a Quality Measurement Management Tool (QMT) to support quality activities and processes. This tool is based on a measurement meta-model repository for collecting, storing, analyzing, and re-

porting measurement data. It is important to note that the proposed QMT supports the IEEE Standard 1012 for Software Verification and Validation (management, acquisition, supply, development, operation, and maintenance) as well as the measurement information needs for Capability Maturity Model Integration (CMMI) processes and products requirements. The proposed repository is generic, flexible, and integrated, supporting a dynamic measurement system. It was originally designed to support Ericsson Research Canada's business information needs.

Software process assessments have become commonplace in the software industry, because the software industry usually does not recognize the level of their software processes. As long as software has existed, the phenomenon of the software crisis has existed, subsuming faulty schedules, inaccurate cost estimates, low productivity of software engineers, and low productivity in general. Promising approaches to the path out of this crisis are developing in the software engineering community. One of the approaches utilizes Software Process Assessment (SPA). **Chapter 5** presents the authors' experiences implementing internal software process assessment at a mid-size Information Technology (IT) company using a customized SPA method. The customized model is based on the Standard CMMI Appraisal Method for Process Improvement (SCAMPI). Software process assessment can be extremely beneficial to an organization with respect to various standards because a smart organization can use the assessment as a framework to evaluate how projects are completed by conscious analysis rather than slavish adherence; thus, an organization can plan and take steps to improve its operations. However, the authors are concerned that the focus of existing assessment processes, at least in the way that many people apply them, tends to over-value technique at the expense of the goal.

Several companies have implemented software process improvement projects. However, some of them give up before the project ends, and others take much longer than expected to complete the project. Therefore, identifying the resistance factors that influence the implementation of such projects might serve, on the one hand, as a reference for professionals in the area and, on the other hand, help manage future projects through the use of preventive actions that either lessen or eliminate the consequences of the resistance factors. To this end, **Chapter 6** presents a survey of 36 professionals involved in software processes improvement initiatives in 18 companies in the state of Rio Grande do Sul, Brazil. The author expected that the results of this survey would contribute to the planning of future software processes improvement projects, in which preventive actions could be designed to lessen or eliminate the consequences of the resistance factors that impede implementation of these projects. In any case, an initial analysis of the risk of these factors is of paramount importance, as is their impact and the probability that associated risks may occur vary across organizations.

Chapter 7 provides reviews of previous literature on the implementation of the Personal Software Process (PSP) in academic settings. The PSP is a structured software development framework that includes defined operations, measurements, and analysis techniques designed to help software engineers understand and build on their personal skills and improve their performance. The PSP provides a specific personal process, guides software engineers in collecting and recording data, and defines ways of analyzing data and making process improvements. Lessons learned from PSP implementations are then highlighted. The authors encountered mixed outcomes due to several issues that later become barriers to the adoption of the PSP. Several tools have been developed to overcome these barriers and to provide a helping hand to students and engineers adopting the PSP.

Software engineering is progressively becoming a collaborative activity, relying on the knowledge, expertise, and experience of a sizable and frequently varied group of individuals. Although individu-

als can develop some software products, it is no longer practical for one person to do most software development jobs because the scale and the complexity of the systems have increased, and the demand for short time to delivery is high. System development is a team activity, and the quality of software products is largely determined by the effectiveness of the team coordination. Thus, **Chapter 8** brings the readers from the individual-focused level of PSP to the team-focused level of the Team Software Process (TSP). The TSP was designed to provide an operational framework for establishing an effective team environment and guiding engineering teams in their work. This chapter provides an overview of the TSP and its associated structures and processes. It also highlights the ways that the TSP operational framework can help project managers and software development teams to deliver successful projects by controlling and minimizing the most common software failure factors. A comparative analysis of the TSP and conventional project management strategies is also presented. Additionally, the results of TSP implementation in industrial settings are highlighted with particular reference to scheduling, quality, and productivity. The last section indicates the additional advantages of TSP and comments on the future of TSP in global software development projects.

From team-focused level TSP, we bring the readers to software process improvement at the organizational-focus level, targeting small and very small enterprises. Although Very Small Enterprises (VSEs) have several beneficial characteristics, such as flexibility and ease of communications, initiating an assessment and improvement process based on well-known Software Process Improvement (SPI) models, such as Capability Maturity Model Integration (CMMI) and ISO 15504 or Software Process Improvement and Capability Determination (SPICE), is more challenging in VSEs. Accordingly, researchers and practitioners have designed a few assessment methods to meet VSEs' needs as they attempt to initiate an SPI process. **Chapter 9** discusses the assessment and improvement process in the context of VSEs after first examining VSEs' particular characteristics and problems. Next, it reviews and considers ways to compare the different assessment methods and standards designed to fit the needs of such organizations. Finally, it presents future research plans on the topic.

In the PSP and TSP practices, data collection and analysis of software metrics need to be performed at fine-grained levels. These tasks are not trivial and require tool support. **Chapter 10** discusses automated tool support for PSP and TSP. The chapter focuses on the issues involved in building such a tool and introduces the authors' ongoing endeavors to develop an integrated tool to support PSP and TSP. Particular attention is given to the features of sensor-based automated data collection for PSP, the utilization of Six Sigma techniques in PSP and TSP activities, and the incorporation of electronic process guides.

The **final chapter (Chapter 11)** of the book describes the benefits of the software process improvement. The Capability Maturity Model (CMM) and the Capability Maturity Model Integration (CMMI) are briefly surveyed and extensively discussed. Prior literature on the benefits and impact of CMM- and CMMI-based software process improvement is highlighted. Much has been learned about the effect and benefits of CMM- and CMMI-based software process improvement from the literature reviewed in this final chapter. There is substantial evidence that software process improvement based on CMM and CMMI can result in considerable improvements in cost, schedule, return on investment (ROI), product quality, and other performance measures.

The primary target audience for this book includes academicians, researchers, scholars, and students who are interested in software process improvement research and related issues. It will also be useful

for professionals who are involved in software development and software process improvement, such as project managers, process champions, quality specialists, software engineers, et cetera. There are no prerequisites necessary to understand the content of this book. It has been organized into self-contained chapters to provide the greatest reading flexibility.

Shukor Sanim Mohd Fauzi
Universiti Teknologi Mara, Malaysia

Mohd Hairul Nizam Md Nasir
University of Malaya, Malaysia

Nuraminah Ramli
Universiti Pendidikan Sultan Idris, Malaysia

Shamsul Sahibuddin
Universiti Teknologi Malaysia, Malaysia