Preface

Software plays an ever increasing role in society today. In fact, software is a factor in almost all aspects of life including health care, entertainment, transportation, communications, and finance, among others. Our dependency on software today is such that the spread of a computer virus can bring our way of life to a standstill for a significant period of time. Demand for an increasing number of software professionals has been cited in business and government circles for at least two decades, with no leveling in sight. In addition, the methods, approaches and tools used to produce the software on which we so much depend are also undergoing rapid expansion. As a result, academic institutions are facing an increasing pressure to produce a greater number of students that are competent software developers.

Software engineering and software development education currently face many challenges. The ever expanding area of software engineering knowledge makes educating the next generation of software engineers a challenge. In addition, the current generation of students has very different interaction patterns than prior generations, making assessing learning difficult. The increasing role that software plays in our lives today (e.g., grid computing, ubiquitous computing, wearable computing, bioinformatics, etc.) requires educators to adapt their education coverage to include these new applications. In addition, many academic institutions must face these challenges within the constraints of program accreditations, university mission, demographics, and even political environments. Clearly, identifying successful approaches to handling these problems is essential to aid software engineering educators. This book contains a generous collection of approaches that represent best practice for software engineering education including student assessment and learning, innovative teaching methods, project-based software engineering, professional practice and ethics, curriculum management and certification and licensing.

This book will be useful to both academicians and practitioners. Academic readers will gain an understanding of proven practices used in software engineering education that could be employed at their institutions. Industry readers will benefit from an understanding of the synergies between educational practices and real-world software development. All readers will gain an international view of software engineering education. Educators can use the book as a reference for adopting novel teaching techniques and for improving their teaching across a variety of computing courses.

The book is organized into seven sections that cover student learning and assessment, innovative teaching methods, project-based software engineering, educational technology, curriculum and education management, and professional practice. Below we provide a brief summary of the chapters.

I. Hislop
In order to provide context for the remainder of the book, the introductory chapter by Dr. Hislop provides an overview of the history and current state of software engineering education. Software engineering is a relatively new discipline and software engineering education is even newer. Dr. Hislop
discusses the issues that have shaped the development of software engineering education including the genesis of a new discipline, the organizational location of software engineering, licensing, certification, and accreditation. The issue of community where software engineering educators can exchange ideas and collaborate is also discussed. In addition, the development and state of curriculum is presented including a discussion of a range of educational venues from entire software engineering programs to individual software engineering courses in other computing degrees.

II. Armarego

An appreciation of learning theory is vital to understanding how best to educate students. In fact, experience has shown that an organized and controlled approach to educating software engineers is more successful than ad hoc approaches. However, the software engineering education community has been slow to explore the application of various learning theories to education. Correctly applied, learning theories could improve the state of software engineering education by allowing educators to design, evaluate and communicate about educational approaches, allowing the best approaches to be identified.

In the first chapter of the section on Student Learning and Assessment, “Constructive Alignment in SE Education: Aligning to What?”, Dr. Armarego argues that learning should fit both the learner as well as the discipline being learned. The chapter explores the alignment between the practices utilized in the software engineering discipline and the models of learning that are used by students to absorb software engineering knowledge in academic institutions. Dr. Armarego discusses the development of a framework for learning that models experiences of software practitioners and suggests that the learning model used for education be characterized and mapped to fit the professional practice. The chapter includes the results of a study which indicates that tailoring the learning models used in academia today holds the potential for improving student software engineering learning.

III. Navarro

Continuing on the role of learning theory in software engineering education, Chapter III “On the Role of Learning Theories in Furthering Software Engineering Education”, by Drs. Navarro and van der Hoek discusses the possible uses of learning theory in software engineering education. This chapter provides a survey of existing learning theories and comments on their use in software engineering education. The authors categorize the current educational approaches in software engineering according to the theories. An example approach which uses an interactive, graphical game to teach software engineering process is used to demonstrate how learning theories can successfully be applied to software engineering education.

IV. Hazzan

The section on Innovative Teaching Methods begins with a discussion of one of the aspects of software engineering that is perhaps most difficult to convey to students, that is, the human perspective including teamwork, conflict resolution, and problem solving from different perspectives. Hazzan and Tomayko present an approach to educating students to the human aspects of software engineering in Chapter IV titled “Tasks in Software Engineering Education: The Case of a Human Aspects of Software Engineering Course”. The approach enhances abstract thinking and expands analysis perspectives of students using a question and task-based approach. The chapter presents a categorization of ten different types of tasks that can be used throughout a course in order to make students more aware of the human-related problems, dilemmas, ethical questions, and other situations that students may face in the software engineering world. The categorization of tasks is based on the authors’ experience in teaching a Human Aspects of Software Engineering course at two different institutions, one located in the United
States and one located in Israel. The chapter presents examples of each category of task and describes the use of the example within a course.

V. Brady

Chapter V titled “Speaking of Software: Case Studies in Software Communication” also addresses the human aspect of software engineering education, specifically both oral and written communication. Typically, communication is given little direct attention in software engineering courses and programs. Teaching communication is difficult and communication in the software workplace is very complex and fraught with subtlety.

Drs. Brady, Seigel, Vosecky, and Wallace are an interdisciplinary team containing both technical writers and software engineering educators that has created an approach to teaching communication skills to software engineering students using case studies. The case studies are based on experiences of software engineering students in a capstone course and are used in the pedagogical sense. These real case studies provide students a complex situation in which to learn and understand communication.

VI. Mead

In this final chapter for Innovative Teaching Methods, titled “Novel Methods of Incorporating Security Requirements Engineering into Software Engineering Courses and Curricula,” Drs. Mead and Shoemaker explore the inclusion of security requirements engineering into software engineering courses and curricula. Security engineering has emerged as a vital national and international concern, part of almost every application designed and developed. These authors explore the integration of security into the earliest stage of the process, namely requirements engineering. The authors identify that security is often considered at either the system level (e.g., authentication, firewalls, etc.) or in isolation from overall system requirements elicitation. To bolster this assertion, the authors provide a careful and detailed analysis of Computing Curricula 2005: The Overview Report, trying to understand the way that security can mesh with the desired outcomes of CC2005. The authors propose and discuss the inclusion of security into curricula, ranging from undergraduate project-based courses to graduate courses on secure systems development to usage of processes such as comprehensive, lightweight application security process (CLASP) and security quality requirements engineering (SQUARE). The bulk of the chapter presents a detailed approach using SQUARE, detailing specific curricula, course content, projects, and so on.

VII. Gary

The fourth section of the book, Project-Based Software Engineering, supports the old undisputable proverb that states that “Experience is the best teacher.” Academics and industry professionals agree that students that graduate with a better understanding of the real world have a better chance of early success in their careers. With the increasing popularity of software engineering course offerings embedded in a variety of computing degrees, inevitably, one must wonder how someone in a classroom could teach students how the real world works. While it is impossible to teach experience, it is possible to teach through experience. This observation has guided the development of many software engineering courses being taught today.

In Chapter VII titled “The Software Enterprise: Preparing Industry-Ready Software Engineers” Arizona State University’s Dr. Gary, describes an innovative approach to learning-by-doing called the Software Enterprise. Under this model, students enroll in two consecutive yearlong software development capstone courses where they learn through experience software engineering’s best practices. At the completion of the software enterprise students have an experiential understanding of how software process can be used to manage the evolution of software artifacts. While this chapter may be quite
helpful to those new to teaching software engineering, due to the interesting way in which the software enterprise brings together so many aspects of the software development lifecycle in two consecutive courses, even experienced instructors may learn a thing or two.

VIII. Roach

In the second chapter for *Project-Based Software Engineering*, titled “Using the Affinity Model in the Capstone Project Course: Teaching Software Engineering in a Computer Science Program,” Drs. Roach and Gates describe their approach for a two-semester software engineering sequence that uses an approach that stresses cooperative (team-based) learning of professional and technical skills. This sequence, underway at University of Texas at El Paso (UTEP), offers a combined two-course sequence taken by students in their final year of study, combining fundamental software engineering topics with the development of communication and team skills, which includes a practical exposure to the software engineering code of ethics and professional practice. Unlike the approach as given in Chapter X, where the capstone project succeeds a much earlier exposure to software engineering principles and practices, this course offered in the Computer Science department at UTEP assumes the opposite – coupling the first exposure of software engineering with the capstone project experience. The authors explore their approach by detailing the curricula, student and faculty responsibilities, project and course requirements, project management, course deliverables, and so on. The authors have evaluated their unique sequence through a combination of surveys that has collected data from alumni and employers; they have had many positive results and feedback. The authors conclude with a discussion of future trends ranging from the high-level (The President’s Council of Advisors on Science and Technology reports on the importance of networking and information technology (NIT) systems and the workforce required to support them) to emerging technology trends (service-oriented architectures) and their impact on curricula.

IX. Klappholz

Clearly, the software industry prefers to hire students who have real-world experience as such students are well-rounded and can more quickly contribute to a project. The presence of an actual client can motivate students and provide direction for a project. However, involving students in projects with real-world clients can pose problems such as locating clients, client communication issues, setting reasonable scope for a project, creating functional teams, assessing the project and more.

Drs. Klappholz, Almstrum, Modesitt, Johnson and Condly present advice for involving students in projects with real clients in Chapter IX “A Framework for Success in Real Projects for Real Clients Courses”. The authors discuss the importance of using real-world projects and present a taxonomy of issues related to involving students in real projects for real clients courses. The authors discuss issues related to client interactions including locating appropriate clients, project-related issues including appropriate projects and scope, team-related issues including team formation and operation, product-related issues including deliverables, and issues related to assessment and evaluation. The approach was developed based on experiences with real-world projects with real customers at a wide variety of U.S. institutions.

X. Demurjian

Continuing in the project area, Drs. Demurjian and Needham discuss the successful and unsuccessful characteristics of a project-based capstone software engineering course in Chapter X, “Experiences in Project-Based Software Engineering: What Works, What Doesn’t”. The authors present the results of 12 combined years of experience in offering project-based courses at two different U.S. institutions. They demonstrate how such courses can be used to support ABET accreditation by providing educa-
tional outcomes assessment. Understanding that obtaining accreditation assessment data can be time consuming, the authors offer guidance to instructors to help manage the assessment data collection. In addition, the authors discuss project attributes and suggest that projects be flexible in order to allow them to be adapted to instructor background. Team size and communication is also addressed and the authors provide a rubric for assessing individual student effort within a team. Future plans include using mixed teams of CS and IT majors.

XI. Bunse

In this first chapter in the Educational Technology section of this book, titled “Applying Blended Learning in an Industrial Context: An Experience Report,” the authors Drs. Bunse Peper, Ochs, Grützner, and Steinbach-Nordmann, explore the usage of blended learning in software engineering education, continuing the investigation of practice-based software engineering in a classroom setting. Blended learning is a technique that combines multiple teaching methods into a single setting, providing a unique perspective and learning experience for students. In this chapter, the authors report on their efforts in blended learning for model-based and object-oriented development with UML, providing an experience which combines self-directed study, collaborative learning, learning with an on-line tutor, social learning, and traditional classroom delivery. The unique aspect of this chapter is that these experiences are related for both an academic and an industrial setting. The work includes a strong case study (questionnaire), data collection, and data analysis of blended learning, offering conclusions based on these results, and exploring future trends such as the use of wikis, podcasts, Weblogs, and virtual learning environments.

XII. Bolanos

Chapter XII, titled “Integrated Software Testing Learning Environment for Training Senior-Level Computer Science Students,” completes the Educational Technology section. Drs. Bolanos and Sierra explore a methodology for software testing that targets senior-level computer science students. The educational technology component in this chapter is to establish an environment that allows actual testing, including: test plans, test case designs, a suite of testing automation tools, analysis and reporting of test results, software configuration management tools (for multiple testing iterations), and a software execution and deployment tool. This is accomplished via an actual, custom, multi-tiered, client server software application developed for this purpose, allowing for a full range of testing (e.g., unit testing, integration testing, functional testing, etc.). As with the prior chapter, the authors rely on a voluntary evaluation survey (93% surveys returned for an average of 150 students taking the course per year) to assess their course, and feed back results into future offerings. In the future, the authors expect constant change, as the underlying development technologies evolve, and more and more testing tools become available.

XIII. McDonald

The history of undergraduate software engineering education in the US reached a critical milestone when the first baccalaureate programs received ABET accreditation in 2001. Since then an increasing number of undergraduate software engineering programs are seeking ABET’s recognition. But for many program leaders, accreditation is still an intimidating event.

For many program leaders and their faculty, a program accreditation exercise goes more or less like this: About a year prior to the accreditation visit, the program leader must first get the faculty to understand why self-assessment should not be an activity that is counted in 5-year cycles but rather, an activity that happens almost daily, and is a natural part of teaching. Then, one lucky faculty member is sent to at least one ABET workshop to learn about accreditation. Upon returning, the terrified faculty member,
now the in-house accreditation expert, calls an emergency meeting no one wants to attend, but everyone attends for fear of being assigned to a laborious (and unfair) accreditation task. The race is then set and faculty and staff rush to collect data and make some sense of it. The accreditation expert earns a couple of course releases to help the program documents in order. About a month or so prior to the visit there are numerous faculty meetings spiced up with incredibly long and fruitless arguments, and endless visits to the department’s copier. The week before the ABET team arrives, tempers run high as the program leader and the accreditation expert put the final touches on what they hope will be a great event.

But preparing for an accreditation visit does not have to be an ordeal. In Chapter XIII, the first in the Curriculum and Education Management section of the book, titled “Software Engineering Accreditation in the United States”, McDonald, Sebern and Vallino explain in simple terms many of the issues involved in an accreditation. The authors cover issues such as making sense of ABET’s criteria, outcomes and objectives, and data collection. One of the most valuable features of this chapter is the way in which the authors, who collectively account for years of experience as program evaluators, program leaders, and in-house accreditation experts; present numerous topics of interest combining their viewpoints succinctly and straightforwardly.

XIV. Ludewig

Software Engineering curricula can resemble vanilla ice cream: they all are called by the same name, but their flavors are quite different. The history of software engineering education is crowded with curricula whose flavors range from strong computer science with nuances of software engineering, to software engineering smeared with heavy blobs of computer science caramel, to the purest unadulterated software engineering. Regardless of their structure, these curricula serve their intended audiences and meet the academic mission of their respective universities.

In Chapter XIV “Software Engineering at Full Scale: A Unique Curriculum”, Dr. Ludewig describes the evolution, content and structure of a software engineering curriculum developed at Universität Stuttgart. The software engineering program Dr. Ludewig describes is somewhat unique in Germany in that, according to his account, no other university in his native Germany has a complete software engineering curriculum. It is based on a principle of individual responsibility and consists of a defined set of initial courses followed by allowing the student great flexibility in the latter courses. In addition, there is only a single set of exams per semester and students are allowed to attend the course in one semester and take the exam for that course the next year or even later.

XV. Rosca

Anyone who has had the opportunity to build an academic program from scratch can identify themselves with the challenges of building the program, and the thrill of seeing its student body grow over the years. While creating a graduate level software engineering program is a formidable task, keeping it up-to-date and maintaining its integrity are essential to ensuring the program’s success over the years. Dedicated faculty must constantly weigh market needs against academic and technical developments such as changes in technology, innovations in software development and maintenance processes, or new software design trends. Then, they must determine how to bring about change to the graduate program.

In Chapter XV titled “Continuous Curriculum Restructuring in a Graduate Software Engineering Program”, Drs. Rosca, Tepfenhart, Wang, and Milewski share with the reader their extensive experience maintaining a master’s level program at Monmouth University over the program’s 21 year history. Due to continuous advances in the engineering of software, the authors assert that maintenance of a software engineering graduate program offers challenges not found in other engineering programs. In addition, the authors discuss their experience maintaining their graduate program factoring in student
input while coexisting with their department’s baccalaureate program in software engineering. Readers of this chapter will benefit from the authors experience maintaining Monmouth University’s graduate software engineering degree over its 21-year history.

**XVI. Frezza**

It was at a NATO conference in 1968 in Garmisch, Germany, where the term Software Engineering was first mentioned in a formal setup. At the time, the term was more a statement of aspiration than a fact. The field of computing as we know it today was still in gestation. Sixteen years later the U.S. Department of Defense awarded Carnegie Mellon University the contract to establish the Software Engineering Institute (SEI) with the intent to “Advance the practice of software engineering because quality software that is produced on schedule and within budget is a critical component of U.S. defense systems.” One of the ways in which the SEI accomplished its mission was to enable universities to develop masters degrees in software engineering. But no one was yet talking of undergraduate degrees in software engineering. It was not until 1996 when the first undergraduate degrees in software engineering were born in the US. Since then an increasing number of schools are taking a serious look at undergraduate software engineering.

In Chapter XVI, Frezza and his colleagues describe in great detail the many issues that surrounded the development of a “Credible Software Engineering Bachelors Program.” The intriguing use of the word “Credible” should spike the reader’s interest in this chapter because, with declining enrollments in computer science in the US and Canada, schools are being tempted with the concept of re-baptizing existing computer science programs as software engineering hoping to capitalize on the upward trend of enrollments in software engineering. As Frezza and his colleagues explain, building a credible undergraduate degree in software engineering requires effort, compromise, and dedication. But more importantly, it requires academic integrity.

**XVII. Thompson**

The final section of the book is titled Professional Practice. Ethics is one important component of the aspect of professional practice for software engineers. The topic of ethics is especially important to software engineering students who will enter a global environment of software development. Upon graduating from an academic program, students must understand their responsibilities with respect to professional practice as well as the role of ethics.

Dr. Thompson addresses the issue of teaching ethics in software engineering education in Chapter XVII, “Ensuring Students Engage with Ethical and Professional Practice Concepts”. Dr. Thompson provides an overview of two widely used codes of ethics, the IFIP Harmonization of Professional Standards and the ACM and IEEE-CS software engineering code of ethics and professional practice. The author then presents an approach to teaching ethics and professional practice in a practical manner which has resulted in increased enthusiasm on the part of students. Dr. Thompson provides insights into effective teaching of ethics including that the education be relevant to the students’ discipline, all instructors should be competent to teach ethics, teaching should respect the values of different people groups and that the teaching of ethics should be pervasive throughout the curriculum.

**XVIII. Seidman**

The final chapter of the book, titled “An International Perspective on Professional Software Engineering Credentials”, supplies an international perspective on professional software engineering credentials. Dr. Seidman provides an overview of forms of credentialing including professional licensing, certification and more. The chapter explains approaches to professional credentialing used world-wide including
broad-based certifications, national examinations, and job frameworks and discusses international efforts to develop standards for these credentials. Dr. Seidman concludes that credentialing software engineering professionals should be distinct from a specific product or tool and that credentialing will become increasingly important as the role of software in society continues to grow.

This book is an aggregation of classroom techniques and experiences garnered from around the world that have been proven successful in educating software engineers. It contains a collection of best practices in the field of software engineering teaching and learning, providing an understanding of the effective educational approaches used in software engineering education. It provides guidance to educators who are already teaching software engineering education or are considering establishing or expanding software engineering education within their institutions. In addition, the book can be used as a resource by software engineering educators to learn and adopt new educational practices to improve education. The diversity of topics and approaches presented provides a broad and international perspective on software engineering education.