Because software engineering is a relatively young field, research in the discipline benefits from the domain knowledge and wisdom accumulated outside its specific realm.

Software engineering is a relatively young field. When the term was first coined in 1968, many other engineering disciplines had already existed for many years. Since then, software has become pervasive in almost every aspect of our modern life, and SE has become increasingly prominent as a highly practical and widely practiced field.

Because of its youth, SE has yet to develop the depth of theory that other disciplines possess; therefore, in many ways, SE research benefits from the domain knowledge and wisdom accumulated outside its realm. The ubiquitous interaction between SE and other scientific and engineering fields is the theme of this collection of articles. However, we do not intend to create another forum for the decades-long debate on whether SE is a true engineering discipline: is SE = E?

THE INFLUENCE OF OTHER DISCIPLINES

What other fields of study have had a major impact on advancing the principles and practices of SE? Since its beginnings in 1968, SE has been provocatively influenced by “engineering” viewpoints. Critical thinkers who are strong advocates for elevating SE into a true engineering discipline have argued that when developing complex, but necessarily reliable, software systems, we must rigorously understand and faithfully apply pertinent engineering principles and practices. Many researchers, such as Harlan Mills and colleagues, who introduced the concept of cleanroom software engineering, and Barry Boehm, who has emphasized the importance of SE process and economic models, have been exemplary in pushing the envelope to the extent that the SE discipline has pragmatically adopted many engineering terminologies and methods.

At the profession level, some may accept the notion that across the traditional engineering landscape we should embrace SE as a newcomer that is a field of intense study. Many other fields of study, while not necessarily regarded as professions per se, have noticeably influenced SE research and practice.

As software includes many facets of human work in a large spectrum of professional and creative activities, SE becomes indispensable to various human undertakings, including petroleum engineering, aerospace engineering, automotive engineering, space exploration, climate control, environmental protection, national security, smart buildings and cities, finance and economics, and healthcare. In these substantial human endeavors, software is increasingly the dominant controller, with widespread proliferation; many such systems require high reliability and are mission or life critical. It is a common belief that SE must assume its bona fide duty to tame complex work that could result in unaffordable or disastrous failures. However, many of us are convinced that SE is not a panacea.

IN THIS ISSUE

Because of space limitations in this theme issue, we can only provide a sample of a small fraction of the computing world to inform Computer’s readers about where and how SE meets other fields. Specifically, authors who are well known in SE have contributed articles to explain the interactions between SE and a very fundamental issue—the theory that sustains SE study—as well as open source
software (OSS), evolutionary computation (EC), space exploration, and services and cloud computing.

In “Can Practitioners Neglect Theory and Theoreticians Neglect Practice?,” Manfred Broy points out that our goal is to develop and evolve “great software.” But what makes software “great?” Many development methods and approaches are based on good principles, but without a theory, it is impossible to evaluate them for qualities such as efficiency and effectiveness. Broy argues that no engineering discipline can hold without an underlying theory, and software engineering needs theory too. He emphasizes that theory does not equate with the use of formalisms, which many developers have found to be impractical.

Brian Fitzgerald considers the seeming contradiction that open source software is popular because it is considered to be of high quality, delivered quickly and for free, using the best developers, yet it seems to breach all of the tenets of modern software engineering. Nevertheless, in “Open Source Software: Lessons from and for Software Engineering,” Fitzgerald points out that OSS benefits greatly from two important SE principles—modularity (information hiding) and configuration management. He also describes how many of the important areas in modern SE originated in OSS.

Mark Harman is well-known for having coined the term search-based software engineering. Yuanyuan Zhang, a member of Harman’s research group, maintains an open online repository that preserves and makes available prior and current SBSE research exploring the use of evolutionary computation techniques in the study of SE (http://crestweb.cs.ucl.ac.uk/resources/sbse_repository). In “Software Engineering Meets Evolutionary Computation,” Harman provides a broad-based yet insightful assessment of the significant influence of EC on SE. Those curious to know how EC interacted with SE research in the past two decades as well as how EC might continue to find its applications in SE study will find much useful information in Harman’s contribution to this theme issue. As he suggests, and we concur, optimization issues are inherent in many SE problems, and SE professionals definitely should not overlook advances in EC.

When a spacecraft is light-minutes or light-hours distant from us, its utility and survival must depend on the software that controls it. Such software must be reliable, updatable, and adaptable to a changing environment in situations where humans cannot provide real-time control. In “Software Engineering for Space Exploration,” Robyn Lutz addresses the role that SE played and continues to play in the development of such software. Not only must the software be reliable, able to operate autonomously in real-time, and adaptable to exceptional conditions, it must also be maintainable over many years as the human team maintaining it changes and evolves. The SE process that NASA and other organizations use must preserve critical knowledge about the software over its often decades-long lifetime while ensuring that the software meets a space mission’s critical requirements.

In “Software Engineering Meets Services and Cloud Computing,” Stephen S. Yau and Ho G. An provide a concise introduction to the intersection issues between SE and these two computing paradigms. This article encapsulates numerous key concepts in services and cloud computing, and it enumerates the most critical, fast-emerging SE challenges that our readers are acutely facing today.

Finally, in “Software Engineering—Missing in Action: A Personal Perspective,” David Lorge Parnas offers his view of what engineering and science are and shares his observations concerning why SE is not yet an engineering profession. Parnas envisions what SE needs to do to become an engineering discipline and notes some of the current impediments to progress in achieving this vision, offering examples from the other articles in this issue.

We are enthusiastic about this collection of articles on a profound subject, and we anticipate that readers will also find them both informative and interesting. The reality is that SE has become a highly sought-after career, and the debate about SE ≠ E is necessarily a key concern to many SE professionals. In the current job market, analysts often label SE as one of the top professions for college students to consider as a career. As educators, we hope that future SE graduates will have the opportunity to read the articles included in this theme issue, and that doing so will help them understand and appreciate the interaction issues between SE and other fields.

References


Carl K. Chang, a professor and chair of the Department of Computer Science at Iowa State University, is Computer’s editor in chief emeritus. Contact him at chang@iastate.edu.

David M. Weiss is the Oanh and Lanh Nguyen Professor of Software Engineering in the Department of Computer Science at Iowa State University. Contact him at weiss@mail.iastate.edu.

Mike Hinchey is the director of Lero—the Irish Software Engineering Research Centre and a professor of software engineering at the University of Limerick, Ireland. Contact him at mike.hinchey@lero.ie.