The term “software engineering” (SE) can be traced back to the title of a 1968 NATO conference. Industry had come to recognize that to create cost-effective solutions to practical problems, scientific knowledge had to be applied to software—that is, software needed to be engineered and not merely crafted. There was little understanding then of how to achieve this goal, though progress was made over the next 20 years. “Software engineering is not yet a true engineering discipline,” Mary Shaw wrote in 1990, “but it has the potential to become one.”

It’s reasonable to ask whether, in 2017, we’ve finally achieved the aspirations of that NATO conference nearly 50 years ago. There are clear signs we’re rapidly moving in that direction. We have codified bodies of knowledge—such as the Guide to the Software Engineering Body of Knowledge (SWEBOK), now in its third edition—and curriculum guidelines for undergraduate SE programs. There are also professional bodies that license software engineers. IEEE and the National Council of Examiners for Engineering and Surveying (NCEES), for example, offer professional licenses in both computer engineering and SE. In addition, the number of accredited SE programs is rapidly increasing worldwide.

Arguably, software is changing more quickly and dealing with more complex problems than any other engineering discipline. A car in 1980 contained about 50,000 LOC; today’s cars contain tens of millions of LOC, and high-end vehicles contain hundreds of millions of them. A typical Linux distribution is also hundreds of millions of LOC. Clearly, we’re doing something right in managing this enormous complexity. Decades of work on software abstraction and patterns have helped us create—and gain intellectual control over—systems of ever-increasing complexity. But SE needs to change to meet the challenges of the future. Software is everywhere in the infrastructure that surrounds us, and it affects all of us. For this reason, new dimensions of SE are gaining prominence.

A bridge or a building is typically built to last a century or more, with periodic maintenance, but software changes rapidly—in some cases daily. We’ve become accustomed to the flood of releases of the software that runs our lives: OSs, desktop applications, mobile apps, and utility software (such as virus scanners). In this way, SE differs from most traditional engineering disciplines: software engineers must deal with the consequences of constantly changing requirements and environments. For this reason, release engineering, continuous delivery, and DevOps have become core competencies that software engineers need to master. For example, Amazon is...
reputed to deploy code every 11.7 seconds, and Etsy does over 50 deployments per day; Facebook updates its code at least twice per day. Furthermore, as software increasingly runs more of our world, including the emerging Internet of Things (IoT), two new areas of SE are gaining importance: green SE and social SE. The environmental consequences of software are rapidly growing; for example, datacenters now account for the same amount of greenhouse gases as global aviation. And with our lives centered on smartphones, power consumption and battery life are among the quality attributes engineers must worry about. As software grows in importance and projects grow in size, software engineers need to be concerned with systems’ technical qualities. A number of key technological developments, such as cloud computing, have made the deployment of ever-larger systems feasible. These systems are part of the fabric of our society—they run our power grids, our phones, the Internet, our businesses, and our government. This trend is set to grow dramatically as the IoT expands. To keep all of this continually running, our systems are becoming self-monitoring, adaptive, and self-healing. The challenge of being “always on” also highlights the importance of cybersecurity: as our world increasingly depends on software, the risks of software errors, flaws, or hacks increase correspondingly.

Finally, software engineers must be aware of the sociotechnical ecosystems in which those ever-larger systems are built, maintained, and used. Software is increasingly open source and crowdsourced. Thus, software engineers need to be not just technical leaders—although clearly that’s a necessary condition—but also community shepherds. In addition to technical mastery of architecture, implementation, tools, and technologies, engineers need to acquire soft skills and be able to guide, persuade, negotiate, and work with others in (often global) interdisciplinary teams.

As the importance of software in our world increases, so do the duties, skills, and knowledge required of software engineers. The challenges are there and we, as a community, must rise to meet them.

REFERENCES


RICK KAZMAN is a professor of information technology management at the University of Hawaii and chair of the IEEE Technical Council on Software Engineering. Contact him at kazman@hawaii.edu.