ing natural forces and a dizzying array of non-technical economic, statutory, societal, and logical constraints. Software engineering is similar except that fewer forces involve the natural world.

▸ *Adapting to changing environments.* Most environments that use computing constantly change and expand. With drawn-out acquisition processes for complex software systems, it is not unusual for the system to be obsolete by the time of delivery. What waste! Mastering evolutionary development is the new challenge.[2]

The System

The problems surrounding the six issues listed here are in large measure the consequence of an overly narrow view of the system for which the software engineer is responsible. Although controlled by software, the system is usually a complex combination of software, hardware, and environment.

Platform independence is an ideal of many software systems. It means that the software should work under a choice of operating systems and computing hardware. To achieve this, all the platform-dependent functions are gathered into a platform interface module; then, porting the system to another platform entails only the building of that module for the new platform. Examples of this are the Basic Input-Output System (BIOS) component of operating systems and the Java Virtual Machine (JVM). When this can be achieved, the software engineer is justified in a software-centric view of the system.

But not all software systems are platform independent. A prominent example is the control system for advanced aircraft. The control system is implemented as a distributed system across many processors throughout the structure where they can be close to sensors and control surfaces. Another example is software in any large system that must constantly adapt in a rapidly changing environment. In these cases the characteristics of the hardware, the interconnections, and the environment continually influence the software design. The software engineer must either know the system well, or must interact well with someone who does. In such cases adding a system engineer to the team will be very important.

We need to encourage system thinking that embraces hardware and user environment as well as software.

Engineering Team

No matter what process engineers use to achieve their system objectives, they must form and manage an engineering team. Much has been written on this topic. Software engineering curricula are getting better at teaching students how to form and work on effective teams, but many have a long way to go.

Every software team has four important roles to fill. These roles can be spread out among several people.

The software architect gathers the requirements and turns them into specifications, seeks an understanding of the entire system and its trade-offs, and develops an architecture plan for the system and its user interfaces.

The software engineer creates a system that best meets the architecture plan. The engineer identifies and addresses conflicts and constraints missed by the architect, and designs controls and feedbacks to address them. The engineer also designs and oversees tests. The engineer must have the experience and knowledge to design an economical and effective solution with a predictable outcome.

The programmer converts the engineering designs into working, tested code. Programmers are problem-solvers in their own right because they must develop efficient, dependable programs for the design. Moreover, anyone who has been a programmer knows how easy it is to make mistakes and how much time and effort are needed to detect and remove mistakes from code. When the software engineer has provided a good specification, with known exceptions predefined and controls clearly delineated, the programmer can work within a model that makes the job of implementation less error-prone.

The project manager is responsible for coordinating all the parts of the team, meeting the schedules, getting the resources, and staying within budgets. The project manager interfaces with the stakeholders, architects, engineers, and programmers to ensure the project produces value for the stakeholders.

In some cases, as noted previously, a systems engineer will also be needed on the team.

Conclusion

We have not arrived at that point in software engineering practice where we can satisfy all the engineering criteria described in this column. We still need more effective tools, better software engineering education, and wider adoption of the most effective practices. Even more, we need to encourage system thinking that embraces hardware and user environment as well as software.

By understanding the fundamental ideas that link all engineering disciplines, we can recognize how those ideas can contribute to better software production. This will help us construct the engineering reference discipline that Glass tells us is missing from our profession. Let us put this controversy to rest. C

References

1. Denning, P. Computing is a natural science. *Commun. ACM 50*, 7 (July 2007), 13–18.
2. Denning, P., Gunderson, C., and Hayes-Roth, R. Evolutionary system development. *Commun. ACM 51*, 12 (Dec. 2008), 29–31.
3. Denning, P. et al. Computing as a discipline. *Commun. ACM 32*, 1 (Jan. 1989), 9–23.
4. Glass, R., Vessey, I., and Ramesh, V. Research in software engineering: An analysis of the literature. *Information and Software Technology 44*, 8 (2002), 491–506.
5. Riehle, R. An Engineering Context for Software Engineering. Ph.D. thesis, 2008; theses.nps.navy.mil/08Sep_Riehle_PhD.pdf.
6. Riehle, R. Engineering on the surprise continuum: As applied to software practice. *ACM SIGSOFT Software Engineering News 30*, 5 (Sept 2005), 1–6.
7. Sangwan, R., Lin, L-P, and Neill, C. Structural complexity in architecture-centric software. *IEEE Computer* (Mar. 2008), 96–99.
8. Tichy, W. Should computer scientists experiment more? *IEEE Computer* (May 1998), 32–40.

Peter J. Denning (pjd@nps.edu) is the director of the Cebrowski Institute for Information Innovation and Superiority at the Naval Postgraduate School in Monterey, CA, and is a past president of ACM.

Richard Riehle (rdriehle@nps.edu) is a visiting professor at Naval Postgraduate School in Monterey, CA, and is author of numerous articles on software engineering and the popular textbook *Ada Distilled*.