Send letters to Editor ■ software@computer.org ■ fax +1 714 821 4010

Software Engineering Curricula Shortfalls

Regarding *IEEE Software*'s Nov./Dec. 2006 special-issue topic, software engineering curricula continue to miss the mark in several critical areas. First, they still fail to reflect the characteristics of classical ("real") engineering disciplines that warrant these areas' esteem and legal status. Computing and math are both important, but real engineers also learn an overall approach that puts the individual techniques in context and includes learning from practical experience.

Second, software engineering curricula give inadequate attention to three elements critical to software success. First, a system will be no better than its requirements, which are neither a subset of modeling nor mainly a software issue. Software requirements creep largely because they fail to meet the real business requirements—an essential artifact that widely held software engineering models completely misinterpret. Second, testing takes up approximately half the time and effort spent on most software projects, yet it receives nowhere near a proportionate share of these curricula's attention. Finally, software is actually implemented only through an organized development process that someone must manage, including leading and managing the human beings who do the engineering work.

Robin F. Goldsmith
President, Go Pro Management
robin@gopromanagement.com

Mike Lutz and Don Bagert respond:

Although Robin Goldsmith says software engineering curricula fail to reflect classical engineering characteristics, he doesn't state precisely what these characteristics are. Existing baccalaureate programs of which we're aware provide an "overall approach" by progressing through topics in design, process definition and evaluation, measurement, and quality. They offer "practical applications experience" in team-based projects, typically culminating in senior-level engineering projects courses. Indeed, in our experience, software engineering curricula are second to none in emphasizing systems-level issues and life-cycle engineering. Without more specifics from Mr. Goldsmith, it's hard to respond in more detail.

We all agree that requirements are the key to any system's success, and certainly poor requirements elicitation and management is a source of requirements creep. We're less sanguine about the assertion that creep is largely a result of such problems; as the target of much of the effort for system upgrades, software requirements are naturally prone to more rapid and extensive changes than hardware components. Indeed, you could argue that software's primary advantage is its adaptability in the face of system evolution; such adaptability, of course, is also its Achilles' heel.

As for the attention given to testing, process, and management, it's important to recognize that curriculum guidelines necessarily reflect a balancing act among many desiderata, not the least of which is providing some flexibility for both individual programs and the students they enroll. So, the guidelines reflect minimum criteria, which most programs will exceed in several areas, thereby providing a unique perspective on our profession.

It's encouraging that Mr. Goldsmith has such an active interest in software engineering education. We would like to invite him to participate more fully by joining the advisory board of an existing software engineering program, by becoming an ABET program evaluator, or by participating with other industrial representatives in the Computer Society's ongoing curriculum development efforts.

We welcome your letters. Send them to software@computer.org. Include your full name, title, affiliation, and email address. Letters are edited for clarity and space.

0740-7459/07/$20.00 © 2007 IEEE