EDUCATING SOFTWARE PROFESSIONALS

Developing Software Engineers at the C-130J Software Factory
by Richard Conn, pp. 25–29. The education and training needs of avionics software engineers are diverse. Several shortfalls in conventional computer science and engineering education can result in mismatched expectations between new hires and their companies. Students must be prepared to work on teams, communicate, and understand the importance of a disciplined, measured, and mature engineering process.

A Collaborative Program to Retrain Lockheed Martin Aero Engineers
by Jorge Díaz-Herrera, Mike Murphy, and Dawn Ramsey, pp. 30–34. This program turns traditionally trained aerospace engineers into software engineers through 30 weeks of university coursework and a company practicum. The lessons learned are not unique; for instance, what works on one project can’t necessarily be replicated.

Software Engineering Programs: Dispelling the Myths and Misconceptions
by Hossein Saiedian, Donald J. Bagert, and Nancy R. Mead, pp. 35–41. New software engineering programs are popular but controversial, and they have been the subject of much hype. This article addresses the academic considerations involved in developing software engineering programs, describing seven common myths about software engineering and shedding light on the tradeoffs and decisions required.

Teaching the PSP: Challenges and Lessons Learned
by Jürgen Börstler, David Carrington, Gregory W. Hislop, Susan Lisack, Keith Olson, and Laurie Williams, pp. 42–48. Software engineering educators need to provide environments where students learn about the size and complexity of modern software systems and the techniques available for managing these difficulties. Five universities used the Personal Software Process to teach software engineering concepts in a variety of contexts.

Simulating a Software Engineering Apprenticeship
by Ken Surendran, Helen Hays, and Andrew Macfarlane, pp. 49–56. By analyzing the prevailing software engineering body of knowledge, the authors established a SE apprenticeship framework, which they use to examine project-based SE courses. In the process, they constructed an enriched model for SE professional development.

Educating Experienced IT Professionals by Addressing Industry’s Needs
by Dale Callahan and Bob Pedigo, pp. 57–62. Academia and industry disagree about what students should learn in school versus on the job. When the University of Alabama at Birmingham developed a graduate program in electrical and computer engineering, they consulted industry executives to develop a curriculum that addresses industry’s needs without compromising academic fundamentals.

Integrating Software Engineering in Introductory Computing Courses
by Grant A. Cheston and Jean-Paul Tremblay, pp. 64–71. An introductory course for undergraduate computing students aims to integrate data structures and software engineering. In addition to completing assignments involving analysis, modeling, and software design, students work in teams using a 10-step process to develop an object-oriented software system.

Teaching Teamwork
by Thomas B. Hilburn and Watts S. Humphrey, pp. 72–77. Few computer science programs offer the software engineering training developers need. Team-based courses are essential to this training and can be structured in various ways. The authors discuss the benefits of using the Introductory Team Software Process in a computing curriculum and discuss its role in preparing students to work as industrial software engineers.

Software Process in the Classroom: The Capstone Project Experience
by David Umphress, T. Dean Hendrix, and James H. Cross, pp. 78–85. A process-oriented perspective on large student projects guides students in integrating end-to-end life-cycle skills and provides consistency of experience among projects. After conducting 49 capstone projects, the authors learned that they must cultivate a process culture, that agile processes provide a bridge from ad hoc programming assignments to organized project work, and that process requires a suitable infrastructure of tools and process expertise.

Teaching Distributed Multidisciplinary Software Development
by Lisa J. Burnell, John W. Priest, and John R. Durrett, pp. 86–93. To create a more realistic distributed collaborative environment, three Texas universities developed an innovative method for teaching collaborative software development in distributed multidisciplinary environments.