The Journal of Systems and Software 82 (2009) 551–552

Contents lists available at ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

Editorial

Software engineering challenges of the "Net" generation

The theme of the 2008 IEEE Conference on Software Engineering Education and Training (CSEE&T'08) was "Educating the Net Generation of Software Engineers." The theme acknowledged the vital role that Internet technologies and applications play in the society and the importance of properly educating and training the "Net" generation of software engineers. The Net generation characterizes the current students who may have never known life without the Internet. Their early and ubiquitous exposure to technology has defined their styles, their modes of communication, their learning preferences, their social choices, and their entertainment preferences. Additionally, the realities of the software industry for which the Net generation need to prepare have shifted from that of the foundational beliefs and practices of many software engineering educators. Thus the educators need to become familiar with the Net era's teaching and training challenges, to investigate the peculiarities of educating and training Net software engineers, and to identify the necessary ingredients for success and for improving our teaching practices and course delivery methodologies.

In addition, Gartner (http://www.gartner.com) has projected the world market for open source software to grow to $35B before the end of this decade. How do we need to change our curriculum to prepare students to develop open source software? The number of security attacks on software is growing exponentially each year. How do we teach students to build security into their software and to implement software consistent with privacy policies? Increasingly, software development teams are geographically distributed as teams are disbursed throughout the globe and as telecommuting is on the rise. Are students learning about communication and coordination practices for working in these distributed teams? As educators, how do we adjust our teaching to meet the personal preferences and technical challenges of the net generation of software engineers? These and similar open issues were addressed at the CSEE&T '08 in a number of keynotes, technical and experience papers, presentations, and workshops. A number of these were selected and further extended for the current issue of the *Journal of Systems and Software*. A summary of each article is provided below.

1. Using Wikis to Support the Net Generation in Improving Knowledge Acquisition in Capstone Projects

In an article entitled "Using Wikis to Support the Net Generation in Improving Knowledge Acquisition in Capstone Projects," Eric Ras and Jörg Rech describe experience factories, which support the packaging and dissemination of valuable experiences in a software organization, provide excellent facilities to support Net Generation students during capstone projects and to create effective learning scenarios. The Net Generation (born 1981–1994) differs from the previous generations in terms of commitment, interaction, and learning style, and hence creates new challenges for education and the usage of technology. The "digital natives" have grown up with technology and are connected, experimental, and very communicative. The authors conduct a small study to find out which Web 2.0 technologies satisfy the needs of the Net Generation. The outcome shows that Wikis match the characteristics of the Net Generation. A Wiki supports asynchronous communication and the collaborative capturing, organization, and distribution of emergent knowledge. Hence, a Wiki was used to implement an experience factory. In addition, this Wiki-based platform called Software Organization Platform (SOP) is able to enrich documented experiences with additional information and to dynamically generate so-called learning spaces. A learning space consists of hypermedia pages structured in such a way that is supports experiential learning. It provides information on software engineering topics according to the selected experience description and current context of the students. These learning spaces intend to cope with the lack of students' background knowledge and explicitly support learning processes. A controlled experiment confirmed that the students acquired about 200% more knowledge when using learning spaces than with conventional experience descriptions. The authors experienced that by using the right Web 2.0 technologies, learning can be made more engaging, social, personalized, and learner-centered.

2. Engaging the net generation with evidence-based software engineering through a community-driven Web database

Although its importance and benefits are increasingly recognized, the awareness and use of evidence-based software engineering (EBSE) is still lower than desired. To popularize EBSE, David Janzen and Jungwoo Ryoo propose a Web accessible repository of EBSE studies. Based on their belief that early EBSE exposure is critical, the authors developed a new course module that serves as an introduction to EBSE early in a software engineering curriculum. Particularly, to appeal to today's Net Generation students, the course module contains, as its centerpiece, a team-based project in which students are required to develop a Web accessible repository of EBSE studies. To successfully complete this project, students must do their own research on dozens of state of the art EBSE studies and summarize their findings in a short descriptive text for each paper they review. Although the quality of their writing carries the most significant weight, student teams also compete with each other on the ideas of Web features that can help galvanize the fledgling online EBSE community consisting of researchers, students, and professionals. Many proven Web 2.0 technologies (such as a user-centric rating system that makes a certain article bubble up or down based on its popularity) have

0164-1212/$ - see front matter © 2009 Elsevier Inc. All rights reserved.
doi:10.1016/j.jss.2009.01.020

already been adopted for the official EBSE site created as a result of the first trial of the proposed course module. The student-generated EBSE study summaries act as the seeds of larger scale, more community-driven discussions on the Net later.

3. Software Engineering Education: How Far We've Come and How Far We Have To Go

The paper written by Nancy Mead, "Software Engineering Education: How Far We've Come and How Far We Have To Go" provides a retrospective view of software engineering education. In this paper Mead discusses the history of software engineering education, some current trends, and a view towards the future. Historic events such as early industrial education and academic degree programs are discussed, along with the players whose leadership led to many early advances in the field of software engineering education. The evolution and growth of degree programs in software engineering education is outlined, along with many professional issues, such as licensing, certification, and ethics. Education initiatives, working groups, and conferences are outlined. The Conference on Software Engineering Education and Training and the Working Group on Software Education and Training are discussed in some detail. She then discusses topics such as industry/university collaboration, development of course curricula and materials, advanced delivery mechanisms, and globalization.

In examining the present, new curriculum efforts, the evolution of educational offerings, as well as the expansion of degree programs and conferences is discussed. Predictions are made for the future of software engineering education, and some of the continuing challenges are discussed. It is noted that many substantive issues remain to be addressed, including recognition of educational research as a valid form of research, the continuing need for mentoring relationships between experienced educators and those new to the field, the need to motivate people to enter the field. The paper concludes that the need for leadership in software engineering education will continue to exist in the future, just as in the past.

4. Single Development Project

Nenad Stankovic presents an article entitled "Single Development Project." The Internet has changed our lives in a profound way, but new opportunities cannot be exploited if not supported by software systems. These, in turn, require software engineers who can work on complex projects and within multiple teams that are supported by many forms of communication. They should possess technical, managerial, and organizational skills. While software engineering courses should bring out the interdisciplinary nature of the discipline, the implementation of student projects must be adjusted to needs of the instructional setting. Students' learning is directly influenced by the kind of roles and tasks in which they engage. New knowledge becomes accessible and relevant when it is supported by problems that place learning within a context. Whenever possible, students should activate prior knowledge and establish a broader understanding of its use and value. In this paper, the author reports on a new project course in software engineering for sophomores, immediate findings, and detail the outcome. Teams of students work within a controllable setting on a broad set of targeted problems that span the software lifecycle. Students use popular methods, current technologies, and tools, while given freedom to define their own goals of achievements, plans, iterative process, and solutions that are appropriate for their level of experience and knowledge.

5. Discovering Vulnerabilities in Control System Human–Machine Interface Software

In "Discovery of Vulnerabilities in Control Systems Human–Machine Interface Software", the authors, Robert McGrew and Ray Vaughn, describe vulnerabilities discovered in a supervisory control and data acquisition (SCADA) software product. These findings are discussed in the context of using these vulnerabilities (and those similar to them) to educate the "net generation" of software engineers about appropriate application of basic secure software engineering principles. The software described in the case study is a human–machine interface (HMI) product, which implements the interface that control panel operators use to view and manipulate components of an automated system, such as an assembly line or a water treatment facility. Many deployments of this software are in areas of national critical infrastructure, making it important to realize the impact of not following secure software engineering practices. Many examples of vulnerabilities involve programming errors (buffer overflows, for example), so it is useful when educating students for them to realize that some vulnerabilities, like the ones disclosed in this paper, are design errors that are introduced in earlier stages of the software development process. The paper also describes how this vulnerability has been used in the classroom to illustrate concepts related to developing software, and procedures used to find vulnerabilities in currently existing software. Suggestions are given for how the case study might be used as an example or a hands-on exercise in several software engineering classes.

Preparation of this special issue has been a learning experience. I'd like to thank the authors for extending their original articles and the reviewers who reviewed the extended papers.

Hossein Saiedian
(Guest Editor)
Electrical Engineering & Computer Science,
University of Kansas,
Lawrence, KS 66045,
United States
E-mail address: saiedian@eecs.ku.edu

Available online 29 January 2009