Software Engineering Education (SEEd)
Peter B. Henderson
Department of Computer Science & Software Engineering
Butler University
Indianapolis, Indiana 46208 USA
phenders@butler.edu

This column was due on July 10, 2004, the date of my son’s wedding [1], so I am a bit behind. Summertime and I will keep this months’ column short and focused.

Learning from Engineering Failures

I am not sure about the rest of you, but as I get older I seem to have more interest in the past and the history channel is one way I learn about and better understand the past [2]. My academic engineering background, and my interests in software engineering and the role mathematics plays in CS and SE education have led to an interest in the historical evolution of mathematics engineering education - learning from the past to help shape the future of software engineering education. In subsequent columns I hope to introduce this historical perspective as I discover more about it. At the same time, I hope the discipline of software engineering is striving to document its history for future generations in the same way that computer science and other science/engineering disciplines are actively doing.

One history channel show that I find mentally stimulating is “Engineering Disasters” and especially the role such disasters have played in both the technical and societal evolution of engineering [3]. We all know of software failures and software engineering educators frequently use these in the classroom to raise student awareness regarding professional responsibility. However, none of these have had the same psychological impact on the public as a bridge collapsing or a space shuttle exploding. Such visual images are extremely powerful.

When software failures do occur they seem to be viewed by the general public as second-order effects – they contribute to, but are not directly responsible for the failure. There are several reasons for this: 1) software is not generally well understood by the public and 2) software failures are often viewed as annoyances and inconveniences, like a pesky mosquito, rather than disasters. This is both a blessing and a curse – a blessing that a major disaster, however the public perceives this, has not occurred due to software errors, and a curse in that there is little pressure from society to ensure the future of this good fortune. I believe that the public does have the perception that the potential for a major disaster is there, but until something actually happens … . This is the “hard” lesson that we should learn from history.

To help sensitize computing students to this I feel it is important that they be exposed to some major engineering failures, software failures [4, 5] and potential failure scenarios. For the latter I often ask students who they would like to have program a heart pace maker for one of their parents. It is important to note that this sense of professional responsibility must be pervasive, that it is more that “ethics” and that one course in ethics, as required in most CS and SE curricula will not suffice[1].

Regarding the education of computing students it is the degree of perserviveness regarding professional responsibility that I have questioned in past SEEd columns [6] and that at least one reader took me to task on this view. Of course, the bottom line is that the ultimate responsibility for engineering failures, disasters, etc. falls on “human” engineers and an important way to help ensure they don’t occur is to properly educate future engineers technologically, socially, ethically and responsibly.

Tree House Analogy

Following in this same theme, I have often used what I call the tree house analogy. The process of designing and building a house is sometimes used to describe the process of developing a software system. Children are fascinated with the possibility of building (engineering) something with their own hands. Most of these are facsimiles of real engineering endeavors which constitute excellent learning experiences. Tree houses are one example.

Software engineering students should not be trained to “build tree houses” (e.g., simply how to use languages, tools, etc.). Unfortunately this is all some computer graduates seem to know or care about. Most students are striving to get the knowledge they need to get a job (HOW) rather than to understand the foundations of the knowledge needed to do the job (WHY). The following familiar quote from Alanis Morissette says this well “The person who knows HOW will always have a job, but the person who knows WHY will always be his/her boss.” Personally I enjoy teaching those students who are interested in “why” and find that they are generally more successful - perhaps you don’t agree with this opinion. This leads to the question of “how” (and “why” **) SMILE **) we should strive to achieve this goal.

Unfortunately, I don’t have a simple answer to this question. Ideally there should be a synergistic relationship between motivating students and learning the fundamental principles of “why.” Also, since CS and SE are relatively new disciplines, it is unclear what the fundamental principles of these disciplines are. In this column and my paper “Mathematical Reasoning in Software Engineering Education”[7] I have stated my view that Computer Science (as a “pure” science), discrete mathematics and logic constitute the key foundational areas of software engineering. In addition, I believe that the fundamental principles need to be introduced early and reinforced throughout the curriculum. Engineering disciplines achieve this through a curriculum requiring foundational science and mathematics courses early.

[1] For example, design flaws can result from technical and ethical failure.
Computing Curricula Software Engineering

The final version of the CCSE Volume is complete and available at http://sites.computer.org/ccse/. This is the first “official” curricula for undergraduate software engineering programs worldwide and I hope you will take some time to look it over. Of course, this volume is just the beginning and over time software engineering will mature into a recognized engineering discipline.

To help colleges and universities that are considering developing undergraduate software engineering programs using the CCSE Volume workshops are being offered in conjunction with various computing conferences. Workshops entitled “Generating Undergraduate Software Engineering Programs Using the Computing Curricula Software Engineering Volume” were given at CSEE&T 2004 and ITiCSE 2004. A similar workshop will be offered at FIE 2004 in October 2004 and at other venues to be announced. If you, or someone you know associated with an institution of higher education are interested in starting an undergraduate software engineering program, please email me and I will put you in contact with the appropriate parties.

For Fun
Try googling “Peter Henderson” SEEd

References