What Should Students Learn in Their First (and Often Only) Software Engineering Course?

James Vallino
Department of Software Engineering
Rochester Institute of Technology, Rochester, NY, USA
J.Vallino@se.rit.edu

Abstract

The questions that are proposed as the basis for academy panel sessions are important ones to ask. In the aggregate, they cover an enormous expanse of the software engineering landscape. The reality of undergraduate computing education is that the vast majority of students do not go through software engineering curricula where there is time to address the academy questions in some depth. Instead, they are in computer science or computer engineering programs, and receive their software engineering education in a single course. What students should really learn in this first, and often only, software engineering course is important because the majority of computing students will not see any other software engineering. The course designer will need to make judicious choices in selecting the material for this course because all of software engineering will not fit in just one course even if you try by using one of the classic software engineering tomes as the textbook. I do not know the right answer to the question I pose in the title of this position paper. I suspect that there is no one set of software engineering topics that should be included, but rather a range of topics to select from based on the purpose and perspective of the course. The answer to this question is important to everyone who has responsibility for providing the software engineering education for the next generation of computing students.

1. Introduction

The Call for Papers asks authors of Academy Panel Session position papers to consider a broad definition of "students of software engineering." At the risk of violating that request, I will narrow the focus of this position paper to undergraduate computing students. The number of these students in software engineering programs will be small compared to the number of the students in computer science, computer engineering, and other computing programs. The reality in curricula for the non-software engineering programs is that all of a student's knowledge of software engineering will be learned in the one, and often only, required software engineering course. I have been at workshops and listened to computer science faculty lament about the struggle of covering all the software engineering topics a student should learn along with teamwork and communications skills in this course. It is nuts to think that you can cram that much learning into one course! Using one of the classic, encyclopedic software engineering textbooks may seduce you into thinking that it can all fit, but it cannot.

2. The tale of an introduction to software engineering course

Of all the courses in our software engineering curriculum, our introduction to software engineering course is the one course that we never feel we have done correctly. The students take the course in their second year after a first-year computer science sequence. This is perhaps a bit earlier in our students' program than at other institutions. The course has a

978-1-4673-5140-9/13 © 2013 IEEE CSEE&T 2013, San Francisco, CA, USA

broader constituency than our department's other software engineering courses because it is also required in the computer science, computer engineering, and computational math programs. In the course's seventeen year history, we have reworked it seven or eight times. We have had versions that used the Personal Software Process (did we get pushback on that) to a full-on agile methodology[1] (it seemed like the students were not learning enough); were heavy on process with less than a 1,000 lines of code in the project (the course was nicknamed Document Engineering) to minimal process, heavy design, and >10,000 line project implementations (the last three weeks of the term were just a hack fest); used no textbook and our own lecture notes (that was a lot of preparation work) to a classic software engineering tome as the textbook (which only half the students bought and none of them read); had teams create full use case requirements, specification, design, and test plan documents (Document Engineering again) to preparing user stories, design, and simple acceptance test specifications (this actually seems like about the right mix); it has been delivered blended[2] and completely face-to-face. The question posed in the title of this position paper is one that we really would like to answer once and for all, if possible. We do not want to continue reworking this course every two years.

3. Selecting topics for the software engineering course

Where can we look for guidance on selecting topics for this introduction to software engineering course? We can look in the computing curriculum guidelines[3] developed by the IEEE and ACM. Software engineering is mentioned as required and elective topics in the computer science, computer engineering, and software engineering volumes. The guidelines are given in hours of coverage for specific knowledge areas along with associated topic lists. Some of the topic lists I can barely read in the time allocated for the knowledge area. In the Software Engineering 2004 curriculum, SE201 Introduction to Software Engineering has 12 teaching modules listed, many of which encompass entire courses, or at least several weeks, in later courses in our software engineering program. You have to question whether it is worth "covering" all these topics because it can be, at most, a surface coverage with little more than buzzwords really learned. Similar coverage is described in the computer engineering and computer science curriculum guides.

You could also look to the textbooks for guidance on course topic selection. The classic software engineering texts will lead you to the same dense topic list with much more material than will fit in a single course. In one of the incarnations of our introductory course, we choose one of the classic texts with the idea that in the introduction to software engineering course we would do only surface coverage. The software engineering students would keep it through their entire undergraduate experience, and we would be able to use it for reference material in most subsequent courses in our curriculum. That idea turned out to be a non-starter for several reasons. The students, especially the non-software engineering students, did not buy into it, or buy the text, and our faculty realized pretty quickly that "if everyone used the same textbook" was not an approach that would work across our curriculum. Even if you flip the class with students doing significant outside class preparation, you will have only surface coverage if you try to make more than a dent in the table of contents of one of the classic software engineering texts.

The approach that we have taken for selecting topics in our introduction to software engineering course has two tenets: less is better, and for our students in software engineering it is OK if some topics are skipped because they will see the full breadth and depth of software engineering through the rest of their program. The course content is more important

for the students in the other majors because this most likely is the only software engineering that they will see. What are the key software engineering learning outcomes that the course should deliver? Using our two guiding principles, some decisions that we have made about course content include: dropping full-detail use cases for user story specification of requirements, emphasize unit testing more and back off on acceptance testing, and completely eliminating functional specification documents which were typically so bad that we saw no educational value being gained.

I think the discussion of this topic can revolve around what programs want their students to learn in an introduction to software engineering course. Reasonable course design says that there are a limited number of topics that can be covered. Should a course focus in particular areas of software engineering, such as, requirements, process, design, or implementation? Does it need to be the "communication course", the "teamwork course", or both? If it does, then time should be allocated for that, displacing some "hard" software engineering topics. So many possible topics, but not nearly enough time.

4. Conclusions

My intention in proposing an Academy discussion to answer the question, "What should a student learn in the first software engineering course", is rather parochial. I am looking for input from others to create a reasonable set of course topics so that the next time we rework our course, it might stick for more than a year or two. I think that there are benefits available for the longer term also. If we can come up with an answer to the question of what topics should be in this introduction course, a longer range goal might be to develop a software engineering concept inventory for an introductory course. Some of the initial work in science and engineering education related to concept inventories was done in physics with the Force Concept Inventory[4]. Having a validated instrument such as that for introductory software engineering topics could provide researchers in software engineering education with grounded measurements of a student's learning gains. I am not aware of any work like this in software engineering education. If the community gains experience with one concept inventory it may be possible to build others for topics such as software process or object-oriented design.

5. References

[1] Reichlmayr, T, "Enhancing the student project team experience with blended learning techniques", Proceedings of the Frontiers in Education Conference, 2005.

[2] Reichlmayr, T, "The agile approach in an undergraduate software engineering course project", Proceedings of the Frontiers in Education Conference, 2003.

[3] IEEE Computer Society and Association for Computing Machinery, "Software Engineering 2004 – Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering", 2004.

[4] Hestenes, D., Wells, M., and Swackhamer, G. "Force Concept Inventory", The Physics Teacher, v30n3, 1992.