Teaching "Software Engineering 'in the large' at Universities"

Position statement for ICSE '94
Workshop on Software Engineering Education)

Roland T. Mittermeir
Universität Klagenfurt
AUSTRIA / Europe

mittermeir@ifi.uni-klu.ac.at

Teaching Software Engineering concepts within a programming course is relatively easy. If students are sufficiently motivated, they would follow the style of the instructor and proceed accordingly. At least judged from the documented code students turn in for assignements, we get this illusion. The problems in these cources are rather with conceptualizing program development and developing algorithms. But assignments given are usually of a size, where these skills can be fairly well acquired.

Thus, with sufficient emphasis on the parts of instructors, I consider the regular classroom-lab situation as adequate for teaching and learning what needs to be learned in terms of Software Development "in the Small".

Extending the scope to medium sized problems becomes already a nontrivial problem. Courses lecturing on desing and V&V-issues might have labs attached, but there are too many things to be addressed for adequately practicing them. Hence, practice has to be relegated to another course, in our case a projects-course. There students would work in small groups and they have to produce a working prototype or small product of something. ("Something" has to be of about the size and complexity that about three students, having had more or less the full undergraduate program done, could solve it from scratch within one semester).

This project-course, which is not taught in class but by consultation and counseling, helps to better understand some of the concepts learned before. But it is still a kind of artificial situation. The teammates are fellow students, the "customer" is also the advisor, and even if there may be more than but one customer, the professor directing the course has always the chance to intervene and to shift between product goals and educational goals. Further, it has to be decided to follow one particular track with any team. Hence, it is hard to explore alternative stiles and procedures.

Therefore, I consider Software Engineering concepts relevant at the team level to be teachable only within certain limits. The gap between believing and experiencing becomes quite often too large and hence hinders the educational purpose.

But if we run into such problems with "Medium Scale Software Engineering Concepts", how then to address issues of larger scale? Those involved with process management? Those involved with inter-team-coordination? One answer would certainly be to ignore them. Universities need not teach everything! There are aspects reality teaches much better than any school. Hence, under the assumption that reality - here in terms of software developing industry - is up to do the job, let's leave it to them.

However, we felt that a large portion of the industrial environment we currently have would not be at sufficiently high a level of development, so that we could delegate this educational obligation without risking to be blamed for neglicence of important aspects. Hence, we do have also a course focussing on such large scale software engineering issues in our curriculum and I was the one to teach it during recent years.

Fortunately, we have another peculiarity in our program. Students are required to spend one semester in an organization, where they should apply what they have learned in school and then return to school to complete their studies in a hopefully well focussed way [1]. This extramural semester provides some help. It gives students not only the perspective of how "real life" is. Due to guided experience reporting, a certain form of experience sharing happens and this allows them to develop a background onto which they can map what they read in the literature and heared in class. Moreover, this (varied) background allows them also to reach interpretations of what is reported in the literature, even if this is - due to differences of scale - not directly applicable.

To reach the target of this "Software-Development-Process Course" and to reap the benefits of the background obtained during the full curriculum and the extramural semester (which is usually taken about one year before the final exams), I give this course to a very large extent into the hands of the students. Its contents is split into two portions:

- 2/3 "What you always wanted to know about Software Engineering, but could not find out so far"
- 1/3 Process Maturity and Quality Improvement

The "What you always wanted to know .." part is conducted by students in a seminar like stile. It is them, who propose the topics (I reserve myself only the right to veto, but this right is very rarely exercised). It is presentation oriented (written material to be provided need not be of the formal quality of a seminar paper, but good enough as handout for the classmates). But the presentations are not monodirectional. It is the group, me, and the other participants of the class, who discuss the problems to be addressed in as open a stile as possible.

The other third is my turn. It is classical teaching about software-process related issues. But as much as I try to help students in their presentations, I let myself help by them during my lecturing in posing questions such as: "How much of this have you seen while working outside"? "Did you see evidence / counter evidence for this"? "How was this problem resolved in your situation"? Hwo would you downscale this proposal to make it applicable to your organization"?

Obviously, as with most real Software Engineering Problems, there is more than one answer to these questions. And if there is none, one can at least pose the question: "How would you transfer this advice into your organization"? Occasionally, case studies are used to give them some more meet to chew. But most of the time, their real experience provides the best cases.

While this software process centered course is still a university course, leading to an examination and plagued with all other problems one might have in classroom situations, we consider it a feasible alternative for teaching what is hard to be taught in a classroom. Finally, teaching it that way, we don’t want to teach only our students. We want them to teach reality after they left us.

Reference:

[1] R. Mittermeir: "Bidirectional Technoloty Transfer in Software Engineering Education"; Proc. 2nd Maghrebian Conf on Software Engineering and Artificial Intelligence, Gammarth, 1992.