Courses for Software Professionals as Two-way Communication Channels between Academia and Industry

Sandro Morasca
Dipartimento di Scienze della Cultura, Politiche e dell’Informazione
Università degli Studi dell’Insubria
Via Valleggio 11, I-22100 Como, Italy
sandro.morasca@uninsubria.it

ABSTRACT
This paper summarizes some of the lessons learned by the author in teaching courses for software professionals. These classes’ primary objective is to provide participants with knowledge about some software engineering methods and techniques. We summarize some advice on how to build and deliver these classes. In addition, these courses are also a good two-way communication channel between academia and industry, since the instructors can get good insights into current practices and future trends. So, these courses help bridge the gap between Academia and Industry in both education and research.

Categories and Subject Descriptors
K.3.2 [Computer and Information Science Education]: Computer science education; K.6 [Management of computing and information systems]: Project and People Management—Training; D.2 [Software Engineering]: Management

General Terms
Human Factors

Keywords
Software Industry, Software Professionals

1. INTRODUCTION
There are several ways in which Academia and Industry can interact within the field of software engineering. Either side needs to bridge the gap that exists between them in both education and research, and all interaction opportunities should be used to this end. In this context, it is not uncommon for professors to act as instructors in software engineering classes for industrial organizations. These are not courses taught in university programs, but they are organized by either a software company for its own employees or by a company whose mission is to offer courses on the marketplace to software companies. It is also possible that these courses are organized by a university—outside its usual programs—and offered on the marketplace. These courses are a part of continuous education activities that software professionals periodically carry out, and they differ from usual university courses, as the attendees usually have different backgrounds and motivations than students in university courses; the duration of these courses may be much shorter; the topics may be much more focused, etc. Also, the professor is a teaching consultant.

This paper reflects on the possible interactions that may take place during classes with software professionals, based on the author’s experience. In addition to teaching software engineering classes at the university, the author has taught classes for software professionals in Italy and Switzerland for about 20 years. Even though their primary purpose is to provide industry professionals with knowledge on some specific method or technique, these courses can also be seen as broader two-way interaction opportunities to help bridge the gap between Academia and Industry in both education and research. We summarize the experiences acquired in these courses by providing a (non-exhaustive) list of recommendations on a few issues about how to teach classes for software professionals in Section 2. In addition, we discuss how the software professionals may provide the academic instructors with information that may be useful in the instructor’s education and research activities in Section 3.

2. GETTING THE MESSAGE ACROSS TO THE PARTICIPANTS
It is well known that continuous education is a very important activity for software engineering professionals. The rapid evolution of the field and the novelty of applications and techniques make it necessary for professionals to keep up to date with the most recent advances. Software professionals whose knowledge had not changed in the last ten years would probably be at risk of losing their jobs without many chances of finding new ones. The speed in the field is different from other, more traditional engineering fields, where the basic knowledge acquired by professionals in university courses does not change much over time, and the basic methods and techniques evolve at a slower pace. For instance, a civil or mechanical engineer whose knowledge had not changed in the last ten years would not be as disadvantaged and a software engineer.
We now list a few recommendations on how to establish a communication channel from the instructor to the software professionals.

2.1 Understanding the participants

As a precondition to creating a communication channel, it is important for the instructor to understand the reasons why software professionals attend a software engineering class. Here is a list of possible reasons (see also [1]): they need to be taken into account during the course.

- Software professionals would like to know more about a specific subject because the knowledge will be useful in their professional life.
- They would like to acquire knowledge that will be useful to obtain a different job within the same organization or elsewhere. Sometimes, the professionals are in the process of re-training from a completely different field [1, 2].
- Their companies believe that the class will be useful.
- The professionals want to obtain a degree, and the course will help them reach this goal.

In the author’s experience, the participants with the first motivation have been the ones with whom communication has been most productive and easy.

The participants’ previous knowledge (on the specific topic of the course being taught or software engineering in general) is another important variable. It is not always possible to do some mandatory pre-screening of the participants by indicating mandatory prerequisites. In my experience, participants have had a broad range of previous knowledge: from those who did not satisfy minimal prerequisites to those who already knew the topic and either wanted an organized explanation of it or simply needed an “official” certification of this knowledge. It has been more easily to communicate with participants with above minimal prerequisites who were actively involved in the course and believed the course would give them some value added beyond a certification.

In addition, participants may come from several different industrial organizations or from the same organization. When they all come from the same organization it is clearly easier to make them communicate better with each other and with the instructor too. Otherwise, especially in short courses, the participants will not tend to mingle with each other and it will be more difficult to have a fruitful interchange between them and the instructor too.

It is usually very helpful to have the participants introduce themselves at the beginning of the course, if the number of participants allows. The instructor can get a better feeling of the participants’ background and interests, and especially of what they are working on, as that is usually the participants’ primary interest. It is also important that the instructor ask the participants what they would like to obtain from the course. It is true that several participants may not have a totally clear idea at that point, but some of them probably do, and their requests should be satisfied to the extent possible. So, the instructor can better focus the course and try to teach in a way that makes it possible for the participants to understand the topics of the course and use them in their practice more easily.

If different participants have different interests that will not allow all the topics of interest to be taught, the instructor must find a balance among all these topics, as also explained in Section 2.4.

One thing that may happen is a mismatch between the course and the professionals attending the class. Actually, it is very possible that in the same course a few people may not benefit much from the course. There are several reasons for this: they are not interested in the topic, the class is taught at a level that is too difficult or too simple for them, etc. It would not be realistic anyway to expect that all the students are as interested in all the topics of a class.

2.2 Critical attitude

Academic instructors may sometimes tend to adopt a critical viewpoint towards the subject they explain, because a part of a researcher’s job is to find possible weaknesses in existing techniques and come up with ways to improve the state of the art. This tends to happen more in courses whose topics are more advanced than those taught during university classes, as often happens in classes for software professionals, although several software professionals actually take classes that may focus on basic topics.

This critical attitude may be the right thing to do in some cases, but it also comes with dangers. It may help the participants develop a critical attitude, which is useful in a field like software engineering, where most of the new techniques and methods have been introduced and accepted in a mostly ideological way, sometimes based on some theoretical argumentation, but hardly any experimental evidence. However, the instructor should remember that, just like the “sales pitches” for software engineering methods and techniques, criticisms too must be substantiated by some kind of theoretical or practical evidence.

On the other hand, the specific technique taught may be an improvement over what the participants know and used. A number of years ago, I taught an introductory class on the then-current version of UML, and I tried to provide a balanced view of the advantages and disadvantages of it. The reaction of the participants to UML was very positive, more so than I had anticipated. The reason was that, up to that point, most of them had used natural language to describe and model the systems they developed. An overly critical behavior is not useful in new fields, since one may very well run the risk of teaching the participants that the field is not promising one, and the participants’ industrial organizations may lose opportunities.

In addition, an overly critical approach may make the participants believe that not only a specific technique, but the entire topic they are being taught is not really worth learning, and they are wasting their time. This may even reflect badly on the instructor, who may pass for someone that explains worthless topics and criticizes everything. On top of that, academic instructors also tend to pass for being more theoretical than industrial ones, so an overly critical attitude towards applied techniques may be even more negative. As a consequence, it is more difficult to create a communication channel between the instructor and the participants.

2.3 Use of laboratories and projects

Software professionals (as well as students in college classes) typically want to go to the lab room often, and the importance of project laboratories is obviously well-known for
education in general and has already been perceived and studied for software engineering education and training to adults [1]. However, the teacher must be very careful, because the participants may erroneously believe that they can carry out practical assignments early. A lot of precious education time may be lost and frustration may arise among the participants. It is up to the instructor to understand when it is right to take the participants to the lab to carry out practical projects. It is true, as [1] says “Make it interesting,” however, this cannot be done to at the expense of providing contents to the participants. This is a clear problem in courses where the participants do not have a strong background in computer science in general and software in particular. I was one of the organizers and instructors of a Web design course in which the participants had to receive a minimum number of lab hours. All of the participants had a college degree, but few from computer science or computer engineering schools. Even though the participants wanted to start creating Web sites as soon as possible, it was soon clear that many class hours had to be devoted to “theoretical” topics before the participants could usefully start building Web sites. So, some of the lab classes had to be actually carried out as “theoretical” ones.

If the projects are carried out in teams, it is important to understand the possible team dynamics. For instance, if professionals at different levels of the corporate ladder are grouped in the same team, it is unlikely that the professionals at the lower levels will contradict their bosses even in these exercises. So, the teacher must be careful enough to mix and match the professionals in the different teams.

It is important that the exercises do not have a unique correct answer, in many cases. Software professionals do not need to feel uncomfortable if they do not provide the “correct” answer—they are supposed to learn in the course—, but there is a pedagogical reason for this as well. In a software quality class, this author usually gives an exercise that, on the surface, looks like a simple counting exercise for software metrics, like Halstead’s Software Science or McCabe’s Cyclomatic Complexity. However, the purpose of the exercise is to make the participants “fill in the blanks,” i.e., complete the definitions of those metrics for the specific languages and programs provided. Thus, there is no real “correct” answer for the exercise, and the software professional get a chance to develop a critical attitude towards these metrics and assess their strengths and weaknesses.

### 2.4 Depth vs. breadth

This is a typical dilemma for all courses, as there is never enough time to cover all the topics deeply enough. Software professionals seem to prefer depth instead of breadth, with the idea that they will know enough of a few topics, instead of knowing little about many topics. This is true even for those participants who are not totally interested in the topics that get explained and that would be more interested in other topics, because they still feel that there would not be enough time to explain those other topics in detail. The teacher must make sure that all really relevant topics are addressed, even though some will not receive as much coverage as the others. In a course about Object-Oriented Software Engineering, one of the requirements for attending the class was that the students knew an OO programming language, and it was clearly stated that the lab exercises would entail the generation and the testing of Java code.

The program of the course also included an explanation of Java and especially of the new features of Java 5.0. This author asked the participants how good their knowledge of Java was, and they said that they did not need to have any classes of Java. There were several other topics, so Java did not get explained. However, through the exercises and the examples, several participants realized that their knowledge of Java was not as good as they thought. However, this realization came too late in the course, when it was too late to provide an adequate explanation of Java.

### 2.5 Flexibility

The instructor needs to be prepared to explain even subjects that he or she did not intend to explain in the class. For instance, the author of this paper was once explaining Object-Oriented metrics to find out that several participants did not know much or anything about Object-Orientation. There would have been little point in continuing the explanation about OO metrics, so it was necessary to make a small break and retrieve at least an introduction to OO fundamental concepts. In another course on UML, a participant told me that it would have been very important for him to know more about testing of OO systems in general. I asked the other participants if it was OK with them to introduce this new topic, and many of them actually had the same problem. So, one of the following classes was on OO software testing, including its relationships with UML.

### 2.6 Using real-life cases from the participants

Instead of using a set of predefined examples, the instructor may use case studies chosen by the participants from their own industrial environments. Here are some possible advantages.

- The participants may be guided to solve a problem they are facing in their professional life. Though it is unlikely that there will be enough time to completely solve the specific problems of the participants, the instructor may provide advice on possible solution avenues which can be beneficial for the participants who described the problem and the other participants as well. Thus, the instructor acts like a consultant.
- The participants learn more about each other’s environments. This is very useful even when the participants come from different organizational units of the same organization, since they have an additional way to interact with each other and exchange information about their own specific environments. In this case, the specific problems of some participants may be the subject of a broader discussion that involves all other participants, who are usually very willing to offer possible solutions, sometimes based on their previous experience. This allows the creation of a very positive and collaborative climate, and the participants have the feeling that they are not taking home just some abstract notions or possibly practical methods that still need to be applied in their own environment, but insights into solving their own problems as well.

One risk is that the participants may choose cases that do not fit well in the explanation of the course, but this is a way to get feedback from the participants on the course itself. If the cases chosen are too far from the topics of the course, some of the explanations may need to be redone.
Also, the instructor may not feel confident in playing on this “foreign” turf, since he or she knows very little about the participants’ industrial environments. It is easier to use predefined examples and exercises, for which the instructor has a solution procedure, but these tend to be a bit artificial or at least to be perceived as such. At any rate, this challenge may yield very positive results and the instructor may have a lot of benefits from this, as illustrated in Section 3.

Another possibility is for the instructor to use a case study from a real-life application developed by the organization for whose benefit the class is taught. It may turn out that some of the participants have even participated in the development of the application or know about it. In a class about software quality routinely organized for the benefit of a company, this author has been using requirements taken from a real-life application developed in the company. In several replications of the course, participants have mentioned that they actually participated in the development of that application and new details are explained to all participants (and the instructor as well).

3. GETTING THE MESSAGE ACROSS FROM INDUSTRIAL PROFESSIONALS

A software engineering professor with the same knowledge as ten years ago would certainly not have adequate knowledge to prepare students for the workplace. So, continuous education should concern software engineering professors even more than software professionals, since professors need to know the state of the art, identify new trends, and do research to improve the state of the art. In this context, software engineering professors need to know current industrial practices, because industrial organizations are their final “customers” in terms of both their educational and research activities: graduates will be hired by the software engineering industry; new needs and new trends continuously arise in the industrial practice.

Given the diversity within the software engineering industry, it is clearly impossible for a professor to have a complete and deep picture about the entire software industry. Nevertheless, software engineering professors should try to use all possible contacts they can have with industrial professionals to get in touch with the new, “hot” techniques, the new developments of existing techniques, or even just to have a good idea of how these techniques are actually applied in industrial organizations.

Courses for software professionals can be an excellent communication channel from the professionals to the professor. A few of the ways used to improve the teaching outlined in Section 2 can be used. The initial round of self-introductions of the participants is very helpful for the professor to understand the real needs of the attendees, which may very well go beyond the lifespan of the course. By having participants propose their own case studies, the instructor may have good insights into the participants’ work environment, practices, methods, techniques, etc.

In this exchange, it is also possible to find that industrial professionals are not using techniques that are believed to be at the forefront of technology. As explained above, Object-Oriented techniques are not necessarily used or known everywhere, especially by professionals who work on legacy systems. Conversely, the professors may receive new information on software development on a number of topics. As an example, in a UML class, this author was made aware by one participant of the existence of a tool that analyzed a natural language specification and semiautomatically produced a UML class diagram. In other cases, variations of existing models and techniques may be better known to some participants than to the professor.

By establishing this dialog, the instructor may have a great opportunity to have better insights into the real issues and problems of industrial organizations. In several classes, when a communication channel and mutual trust are established between the industrial professionals and the instructor, the industrial professionals that participate in a class are usually more open than in other, more formal settings. Even the instructor’s critical attitude may show the participants that the instructor does not have a “product” to sell, but is interested in providing the participants with balanced information. So, they are likely to provide the instructor with information that may help the instructor in his or her professional and research activities. To be clear, the professionals will never disclose any industrial secrets. However, it is information on industrial practices that is not readily available to professors who do not get in touch with industrial professionals. It is only obvious that the instructor must use any piece of information communicated by the industrial professionals in an ethical way.

Evaluations from the participants are very important, and it is not easy to make the participants understand this. Participants may believe that their observations will not be taken into account, or they may just not be interested in providing a feedback after the class is over. It is important to collect evaluations, observations, and suggestions and then discuss them with the participants. These comments can be used by the instructors in their research and education activities, at university and in other replications of the course for software professionals. The data are usually anonymous, but it is not infrequent to obtain questionnaires that are actually signed. This actually shows that a good communication channel has been created and the participants do not feel threatened in any way by speaking their minds. The discussion at the end of the class can be used along with any kind of feedback obtained during the course for improving its future repetitions. Also, the knowledge of the participants’ initial expectations plays a role for the interpretation of their final comments.

4. ACKNOWLEDGMENTS

The research presented in this paper has been partially funded by the IST project “QualiPSo,” sponsored by the EU in the 6th FP, the FIRB project “ARTDECO,” sponsored by the Italian Ministry of Education and University, and the project “La qualità nello sviluppo software,” sponsored by the Università degli Studi dell’Insubria.

5. REFERENCES
