ABSTRACT
Defined software engineering process help teaching and guiding software engineering courses projects. However, using them raises several issues related to process and course features. Architecture issues relate to matching process and course lifecycle models. Size issues address project scope and extent. Support issues deal with student and instructor materials and tools.

Categories and Subject Descriptors
K.3.2 [Computers and Education]: Computer and Information Science Education – information systems education.

General Terms
Management, Measurement, Experimentation, Standardization.

Keywords
Software process, course project, lifecycle model, software engineering education.

1. INTRODUCTION
This contribution concerns software course projects, structured along documented software development processes. We have used one such process for course projects since 2000, mostly with classes of “specialization” programs (in Brazil, a kind of lightweight M.Sc. program, tailored for industry professionals).

Almost all of those students had several years of software development experience, although most of them were not acquainted with up-to-date technology and processes. A few were able to program in Java; knowledge of UML, or SW-CMM and similar models was superficial, in the best cases. Indeed, many came to our courses precisely because they felt they need to update in these subjects.

Here, we focus on the main challenges that we had to address in those courses, and on some solutions found. The base process used in this work is reviewed in the second section. The remaining sections group the issues in three blocks, concerning, respectively, architecture, size and support issues.

2. BASE PROCESS
Process-based course projects enact education–oriented processes, such as those published by Humphrey (PSP, TSP) and Boehm (MBase). Praxis, the process used in the cases described here, was itself described in [3], [4], and [5].

The broad architectural features of Praxis resemble those of the Unified Process (UP) [1], which should not be confused with the RUP, a derived, but quite different, commercial process. Praxis has the same phases and workflows as the Unified Process, but its activities, artifacts and roles are much simpler.

Our department uses a tailored version of this process in industrial development for outside costumers. Many students that take the course projects also work in those industrial projects, although they are completely separated from academic work.

3. ARCHITECTURE ISSUES
Well-defined process steps produce standardized outputs, subject to objective acceptance criteria, at given milestones. In real life, this empowers managers to track and control their projects. For course projects, those milestones and criteria provide good means for assessment of the acquired proficiencies.

Verification and assessment of the output artifacts might be a very heavy workload for the instructor and teaching assistants. We addressed that by using standardized inspections and tests as criteria, and by requiring student teams to inspect the work of their colleagues, recording their findings in standardized reports. The teaching assistants then perform a second-level review of those reports, re-inspecting samples of the output artifacts, much as a quality assurance group would do.

Judging by the widespread use of the RUP and the XP, industry seems to show heavy preference for the spiral lifecycle model. This is also the choice model for TSP, MBase and the UP. It exercises activities of all process disciplines in every phase and iteration (terms as defined in [2]). Therefore, it requires that students start the course projects having theoretical knowledge about all those disciplines, at least.

This is a problem if the project course also happens to be the first or only software engineering course in a program, as has generally happened in our case. This situation would be easier to handle with the waterfall model. However, by means of careful scheduling, it is possible to follow the spiral model from the Construction phase onwards, while having a waterfall-style Inception and a mixed-mode Elaboration. Interestingly, this also happens to be our current preferred lifecycle model for contracts with outside customers, in this case for business reasons.

4. SIZE ISSUES
Size issues relate to exercising a maximum of software engineering practical issues, while fitting the project within given
course schedules. Our experiments with complete course projects used a two-semester (120 lecture hours) schedule. To achieve this, it was necessary to restrict both problem size and number of project iterations.

Generally, we start with a problem measuring circa 100 function points (about 5000 to 7000 Java logical lines of code). This size is kept in the first three iterations, heavily oriented towards requirements, analysis and project planning. The remaining work is performed within a full spiral model, at 20-30 function points per iteration. Currently, the students are able to complete two iterations more, thus achieving complete implementation of about 40-50\% of the requirements.

This has been enough to exercise the most significant features of several standards and practices. During the last years, we have been able to increase project size without extending the schedules, thanks to process improvements designed to promote agility, automation and reuse.

5. SUPPORT ISSUES
For a course project based on a well-defined process, it is necessary to supply the students with relevant process documentation, which might mean a few hundred pages of text. A textbook for these courses has to contain at least this material. We have also found useful to include abridged discussions of the most relevant software engineering techniques, with references pointing to both paper and Web literature, for those who need deeper information. This is especially true for industry students, who seldom have the time to browse the Web at leisure.

On the other hand, while some software engineering techniques remain stable for decades, some very recent process and technology developments may also be relevant. We have found it necessary to keep a support site, where new or updated material may be posted as needed. Also, this site serves as a portal to other sources of software engineering material.

In many schools in Brazil, software course projects are taught by industry instructors, moonlighting as part-time faculty. They are generally experienced in practical issues, but do not have much time for preparing courseware. For those, we have found that availability of instructor materials (slides, exercises and examples) is a major factor in the choice of a software process for course projects.

Finally, there is the question of tools. Many industrial-grade software engineering tools are expensive, and out of reach of most programs. This was partially solved by our process, by casting many artifacts as text documents or spreadsheets. For UML modeling, however, either the instructor must use freeware tools, or she/he must enroll in one of the academic support programs offered by the industry.

6. FINAL REMARKS
According to our experience, teaching a course project based on a defined software process is an effective way to teach relevant software engineering issues and practices, within limited resource and time budgets. We have found this especially useful for recycling industry-based students. We used a process that was designed for this purpose; we expect that improving this process may lead to even more effective course projects.

7. ACKNOWLEDGMENTS
We thank IBM Rational for supporting this work, within the IBM Scholars Program.

8. REFERENCES