ABSTRACT
Software engineering education must provide the students with knowledge and practice of software development processes. These must be used in course projects, to confront the students with realistic software engineering issues. A requirements set for educational software development processes is proposed here. It includes requirements about process architecture, team orientation, project life cycle, standards and practices, student support and instructor support. Some published real-life processes were evaluated against these requirements, and a new process was designed to meet them.

1. INTRODUCTION
We have been teaching software engineering for many years. Our students have come from both undergraduate and graduate academic programs, and also from industrial training programs. We have also mentored many software engineers, in real-life software development projects. As expected, those who had an adequate software engineering background perform much better in meeting commitments and delivering high-quality products. According to our experience, this adequate background must include knowledge and practice of software development processes.

In section 2, we discuss the rationale for an educational software development process. In section 3, this rationale is translated into a set of requirements. In section 4, well-known processes are evaluated against these requirements. Section 5 presents a brief description of a process designed to meet these requirements. Conclusions are drawn in section 6.

2. THE NEED FOR AN EDUCATIONAL SOFTWARE DEVELOPMENT PROCESS
Improving the performance of real software organizations is generally a very difficult task. Even if upper management is willing to sponsor process changes and to pay for expensive consulting, training and tools, hacking-addicted programmers will resist the changes. They may recognize the need for better processes, but are generally afraid that these might delay product deliveries and jeopardize their jobs.

Therefore, teaching students to follow defined software development processes must address a cultural issue. The students must be exposed to industry-level best practices, and they must learn to evaluate them, in order to decide later which will be better choices for their real work environment.

At a practical level, defined software processes may supply a thread for more efficient delivery of knowledge and practice in software engineering courses. If a software engineering course offers disjoint knowledge chunks, it is difficult to assess which kind of proficiency was really acquired by the students. Also, time is generally short to expose the students to a meaningful set of best practices. Especially, industrial-training programs must be very focused, in order to be effective.

On the other hand, software engineering academic programs are not common. Many software developers come from computer science programs, where software engineering is just a discipline among others, which must be squeezed in a few courses (sometimes, just one). It is common to have software engineering as a mandatory course that must be taught in one or two quarters or semesters. Often, this results in having classes with large numbers of students. The instructor workload is compounded by the need to assemble a large diversity of topics into a coherent whole.

If a software engineering course follows the lines of a well-designed software development process, it can expose the students to a consistent set of chosen practices. Course projects may be designed to present real-life issues to the students and to impart basic proficiency levels.

3. A SET OF PROCESS REQUIREMENTS
In this section, we adopt the following definitions, given in [3].

- A process is a set of partially ordered steps intended to reach a goal.
- An enactable process is a process that includes all the elements required for its performance by one or more agents.
- A process architecture is a conceptual framework for consistently incorporating, relating and tailoring process elements into enactable processes.

Additionally, we define an educational process as an enactable process whose architecture and elements are adequate for educational purposes.

We propose here a set of requirements for educational processes for software development. These are summarized in Table 1.

1. Architecture. An educational process should be composed of steps with fully defined entry criteria, inputs, activities, outputs and exit criteria. It should be possible to map the
completing some key steps onto control points. This supplies the instructor with milestones for student assessment and grading, and also helps teaching the students the importance of meeting commitments. On the other hand, an educational process should recognize the fact that all real-life projects are more or less iterative. Good planning, specification and design should be encouraged by the process, but rework should be allowed, when necessary.

2. **Team orientation.** An educational process should be designed for enactment by small teams of students (say, three to five people). In real life, this will be a normal team size for many projects. Individual work severely limits the size of the projects that can be executed, and does not allow for practices that are inherently team-oriented, such as reviews and JAD (Joint Application Development). Larger, multi-tiered teams are much harder to coordinate and assess.

3. **Project cycle time.** An educational process should support the whole life cycle of typical software development projects. Generally, this means a duration ranging from four months to about one year. The process should allow splitting projects that are longer than the duration of a single course, in a meaningful way.

4. **Standards and practices.** An educational process should expose the student to widely recognized standards, practices and paradigms. As it happens with traditional engineering branches, students should be required to employ standard notation and to follow standardized procedures for important activities of the process steps. Examples include notations such as UML (Unified Modeling Language) [2, 11], models such as SW-CMM (Software Capability Maturity Model) [10], specific standards such as those provided by ISO and the IEEE [6], and common practices such as OOT (object-oriented technology), reviews and inspections, JAD workshops, and sizing and estimating methods, such as function points.

5. **Student support.** One or more textbooks should support an educational process. The required standards should be supplied as part of the process materials. Templates and forms ease considerably the adoption of standards. Checklists should be supplied, to help reviewing whether the project artifacts conform to the applicable standards. Required tools should be simple and inexpensive. They should alleviate the most boring tasks and help to weed out trivial defects. On the other hand, sophisticated tools should not be required. Besides being expensive, they generally require too much effort to acquire proficiency. Also, sometimes they hide process details that the students should understand correctly, even if they are going to use state-of-the-art tools later in real life.

6. **Instructor support.** An educational process should contain elements to offer adequate support for the instructor work. These may include, besides student materials, other elements such as slides, examples and exercises.

### 4. EVALUATION OF SOME PROCESSES

In this section, we evaluate some widely recognized processes against the requirement set above proposed. These include PSP and TSP, developed by W. S. Humphrey [3, 4, 5], and the Unified Process, developed by I. Jacobson, J. Rumbaugh and G. Booch [7].

#### Table 1 – Summary of proposed process requirements

<table>
<thead>
<tr>
<th>Area</th>
<th>Requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td>Architecture</td>
<td>Fully defined process steps</td>
</tr>
<tr>
<td></td>
<td>Mapping between process and course milestones</td>
</tr>
<tr>
<td></td>
<td>Allowance for iterative work</td>
</tr>
<tr>
<td>Team orientation</td>
<td>Adequacy to small teams</td>
</tr>
<tr>
<td>Project cycle time</td>
<td>Typically 6 months – 1 year</td>
</tr>
<tr>
<td></td>
<td>Possibility of splitting larger projects between courses</td>
</tr>
<tr>
<td>Standards and practices</td>
<td>Exposure to recognized standards and paradigms (UML, CMM, IEEE, ISO etc.)</td>
</tr>
<tr>
<td></td>
<td>Exposure to widely used industry practices (OOT, JAD, inspections etc.)</td>
</tr>
<tr>
<td>Student support</td>
<td>Textbooks (including standards)</td>
</tr>
<tr>
<td></td>
<td>Templates, forms and checklists</td>
</tr>
<tr>
<td></td>
<td>Simple and inexpensive tools</td>
</tr>
<tr>
<td>Instructor support</td>
<td>Instructor material (slides, examples, exercises etc.)</td>
</tr>
</tbody>
</table>

#### 4.1. PSP and TSP

The PSP (Personal Software Process) is really a series of seven increasingly sophisticated software development processes, meant to be taught at graduate and advanced undergraduate level. These processes are taught through a sequence of small individual projects. A simpler version of PSP, to be used in introductory programming courses, is presented in [4]. The TSP (Team Software Process) provides a natural sequence for PSP. It is described in [5]; actually, this is TSPe, meant to be an educational version of TSP. TSPe is designed for student teams with four or five different roles. The course project is a small application, developed in a few iterations.

In the following paragraphs, we evaluate PSP and TSP as a whole, analyzing them against the proposed requirements.

1. **Architecture.** Both PSP and TSP are rigorously defined. Scripts available in the respective textbooks define the process steps and elements. PSP is mostly a waterfall process; in every step, previous artifacts may be reworked, but rework items are counted as defects. Only PSP3, the last process of the PSP sequence, allows implementation to be performed in cycles. TSP, on the other hand, is intended for cyclic development; within each cycle, it is a waterfall process too.

2. **Team orientation.** PSP is inherently individual. TSP is inherently oriented towards small teams.

3. **Process cycle time.** PSP projects are very small, with typical duration of about 10 hours each. The last project, using PSP3, should be somewhat longer. TSP completes three development cycles in a semester course.

4. **Standards and practices.** The PSP textbook includes some standards required by the process practices. It covers advanced practices, such as those required by SW-CMM levels 4 and 5, but most basic level 2 practices are left out, such as requirements, quality assurance and configuration management. TSP requires the students to use several standards for its main technical activities, but these standards are not supplied as part of the process.
5. **Student support.** Textbooks are available for the full PSP [3], introductory PSP [4] and TSPe [5]. The full PSP textbook contains some standards and checklists. All these textbooks offer a complete set of templates and forms for the respective process versions. Spreadsheet versions of the forms can be downloaded from support sites. No specific tools are required, besides a development environment. However, some groups have developed support tools, to ease data collection and reporting in PSP.

6. **Instructor support.** A number of instructor support materials can be obtained from the PSP textbook publisher. These include a slide set, teaching guidelines, grading checklists and spreadsheets for process data analysis and presentation.

PSP and TSPe, on the whole, offer an excellent example of educational software processes. Indeed, the requirements set proposed here is heavily influenced by our experience with PSP teaching. However, there are some issues to be addressed. We comment here on required student background; a critique of other aspects can be found in [8].

The full PSP requires strong discipline from the students. In our experience, students who had been previously exposed to more traditional software engineering methods and issues were able to appreciate PSP; they fared significantly better. These students had also some practice with object-oriented analysis and design; they were able to use it to advantage, during PSP planning and design exercises. It seems that the full PSP (and, presumably, TSP) gives best results when applied with advanced students, who already know about major software engineering issues, and who have the necessary motivation to employ advanced software processes.

### 4.2. The Unified Process

The Unified Process is a full industrial-strength software development process, proposed by the authors of the Unified Modeling Language. It has quickly earned wide acceptance among software developers. The Rational Unified Process (RUP) is an enactable version of the Unified Process, sold by Rational Corp. The Unified Process is fully described by its textbook [7], while the RUP is embodied in an extensive hypermedia knowledge base, sold by Rational.

Several critiques of these processes have been published; see [1], for instance. Here we consider their possible educational use, analyzing them against the proposed requirements set.

1. **Architecture.** The Unified Process has a kind of matrix architecture. From a management viewpoint, it follows a linear sequence of phases, divided in iterations, that provides sharp control points. From a technical viewpoint, it is a fully iterative process. It has five standard workflows (requirements, analysis, design, implementation and test), which cover most of the technical software engineering activities. There are no managerial workflows; these are supplied by the RUP. The definition of the process steps is rather informal.

2. **Team orientation.** The Unified Process seems to be primarily oriented towards large teams, given the large number of roles described in its workflows.

3. **Process cycle time.** In principle, the Unified Process might be used in educational projects of typical duration, if it is tailored for use by smaller teams.

4. **Standards and practices.** The Unified Process offers very detailed and thorough methods for object-oriented activities, especially analysis and design. It deals with other activities, such as requirements, implementation and tests, in a more superficial way. It does not deal with some usual tasks and techniques, such as project estimation and planning, reviews and inspections, user interface design, user documentation, quality assurance and configuration management. The RUP offers more support for some (but not all) of these issues.

5. **Student support.** The Unified Process and the RUP offer no specific student support, other than the materials offered to normal software engineers. The RUP is expensive for students, and it is best performed with the Rational Suite tools, which are quite expensive.

6. **Instructor support.** Same as above.

In conclusion, the Unified Process seems to be a very strong contender in the field of industry-level processes. For educational purposes, it is too heavy and detailed in some aspects, but leaves out a number of software engineering issues that should be addressed in courses. Even if an organization decides to use the RUP as its standard process, we think the initial training of its workforce should be based on a lighter process, specifically designed to meet training objectives.

### 5. THE DESIGN OF AN EDUCATIONAL SOFTWARE DEVELOPMENT PROCESS

We designed and implemented the Praxis process to meet the above stated requirements, as follows.

1. **Architecture.** The Praxis architecture inherits from the Unified Process sequential phases and iterative workflows (Figure 1). Unlike the UP, the phases are divided in preset iterations, whose completion can be mapped to course assignments. Scripts, similar to those found in PSP and TSP, define the iterations (Figure 2). The workflows correspond to those of the Unified Process (requirements, analysis, design, implementation and test), augmented by project management and quality management, based on the SW-CMM level 2 key practices. The workflows are defined as a partially ordered set of activities, and include practices pertaining to the workflow as a whole (e.g., JAD for requirements). Higher-level SW-CMM practices are not addressed; students are expected to learn them in follow-on courses, such as PSP.

2. **Team orientation.** The process standards were designed for use by small student teams. Currently, there are no mandatory TSP-style role assignments, but the instructor may induce the students to define them.

3. **Process cycle time.** For small projects (typically half a dozen use cases), the students should be able to produce all the artifacts in two semesters, or the most significant ones in one semester.

4. **Standards and practices.** The process artifacts conform to several IEEE standards, such as requirements specifications, project plans, quality assurance plans, design descriptions, test plans and specifications and user documentation (Figure 3). Other standards cover important practices, such as user
interface design, detailed design and coding. UML is used throughout the process, both as required notation for the analysis and design models, and to describe the process elements themselves, as in this paper.

5. **Student support.** The process is fully described in a textbook [9]. The textbook contains the process scripts, a succinct description of a number of recommended practices, and the text of the required standards and the respective checklists. Most artifacts can be built with a word processor or a spreadsheet tool. Analysis and design models require a UML tool. Commercial UML tools are quite expensive, but some vendors are willing to offer quite favorable educational licenses. Open-source tools may become a future alternative.

6. **Instructor support.** A companion instructor CD contains templates and examples, to ease the production of the artifacts. Also, it contains slides and exercises that cover all the topics.

In its current version, the author and a colleague used Praxis to teach two undergraduate classes in 2000; preliminary versions had been used since 1997. These classes had one-semester courses; the students were able to complete the requirements, analysis and design workflow artifacts. We expect to perform soon a two-semester experiment, to assess the feasibility of completing the whole task, implementation and management workflows.

In other experiment, Praxis was used as an individual process, in one M. Sc. dissertation. Here, a graduate student developed a project management tool for Praxis itself, comprising 26 use cases and 340 function points.

Praxis is currently being introduced in several larger real-life projects. The first of these, comprising 67 use cases, 460 classes and 2,400 function points, was completed in 6 months, with a total effort of 60 programmer-months.

6. **CONCLUSIONS**

Educational software development processes address specific needs of software engineering education. People who are just acquiring proficiency in their subject matter must enact them with limited resources, in limited time. On the other hand, these processes must exercise a broad range of significant real life best practices, and expose the students to issues they are likely to meet in their professional life.

We proposed a set of requirements for this kind of processes, and used this set to evaluate some widely recognized processes. This evaluation led to the design of the Praxis process.

7. **REFERENCES**


