YOOPEEDOO (UPEDU): A Process for Teaching Software Process

Pierre N. Robillard, Ph.D., P.Eng
Ecole Polytechnique de Montréal
C.P. 6079, Station Centre-ville
Montréal, QC, Canada, H3C 3A7
Tel: 514-340-4238, Fax: 514 340-3240
pierre-n.robillard@polymtl.ca

Philippe Kruchten, Ph.D., P. Eng.
Rational Software Canada
650 West 41st Avenue, Suite 638
Vancouver, BC, Canada, V5Z 2M9
pbk@rational.com

Patrick d'Astous, Ph.D., P.Eng.
Ecole Polytechnique de Montréal
dastous@info.polymtl.ca

Abstract

Process in software engineering is a growing concern for many software development organizations. The need for well-educated software engineers is bringing new software engineering programs to universities. In many programs, software process education adds up to a few hours of lectures in an introductory software engineering course. This paper presents the structure and the content for a full, one-semester course on software process, which has been designed in close collaboration with industry. The course is based on a software process call UPEDU, and has been customized from the RUP for the educational environment. Many artifacts derived from a project case study are used as examples or templates. The content of the course is oriented towards the cognitive skills needed to perform the various activities required of software process. This material will be published in textbook format with CD by Pearsons in fall 2001.

Keywords: Software process, UPEDU, Software Process Education

1: Introduction

The software process is becoming a major concern in most software development organizations. SPICE, CMM, RUP and OPEN are the new keywords, and process diversity has been stressed in a recent issue of the magazine, IEEE Software [1]. Although industries are relying increasingly on defined process activities in developing their software, very few computer science and computer engineering programs offer a dedicated course on software process. Software process is mainly taught as one subject among many in a first or second course on software engineering [2, 3]. Considering the importance of software process in the industrial environment, we believe that there is a need for a full course in software process in
any software-engineering-oriented program. There are many difficulties associated with designing a course on software process, however. One of these derives from the immaturity of a domain which is just 10 years old [4], and another from the number of differing viewpoints on the meaning of software process.

The first, but highly diffused, expression of interest in software process was manifested in the SEI’s CMM approach [5]. CMM (the Capability Maturity Model for software development) is a model designed to assess the maturity of the software processes of an organization and to identify the key practices required to increase the maturity of these processes. This view is based on assigning practices to one of five maturity levels. The purpose of the model is to ultimately derive a maturity level for the organization. The CMM approach leads naturally to standardization, and is supported by ISO/IEC 15504 - (An Emerging Standard on Software Process Assessment) [6]. The model is straightforward and the various sets of practices are based on common sense and empirical evidence. The course content for approaches such as CMM is purely descriptive, however, and can be taught within a few hours.

The Personal Software Process (PSP) has been defined as a teaching tool for students who are improving their software programming skills [7]. This approach defines a series of improvements to an individual’s development practices to be presented to the student in a specified order. With this approach, personal data collection is stressed, and improvements are suggested based on the way in which the data are interpreted. The PSP structure is designed to fit into a university program and is supported by well-defined exercises. It is very much oriented towards individual improvement at the construction phase of the software life cycle. Such an approach will be difficult to follow for software engineers who are less familiar with coding, like software architects, software managers, software designers and configuration managers, but who are at the same time very much involved in the software process.

Software process is also linked to methodology, either with a lowercase m or an uppercase M [8]. Thus, Methodology concerns everything about how a group repeatedly produces and delivers systems: team, skills, roles, tools, environment, artifacts, techniques, standards, products, etc. The Methodology includes all aspects of software development, and is therefore more comprehensive than the software process. Although software engineering students could benefit a great deal from learning about Methodology, the topic is related more to the organizational aspect of the environment than to software engineering itself. Industrial engineering students are more likely to take an interest in Methodology implementation paradigms. By contrast, a methodology describes a few techniques and drawing notations for a few roles, such as those of software architects and designers. Many university software engineering programs offer courses on these ‘small-m’ methodologies, which are called software design or software analysis courses, and which are sometimes based on formal notations like UML or Z. These methodologies or design processes are usually only a small component of what industries call software process.

Software process could also be defined in conjunction with software life-cycle processes [9, 10]. Life-cycle models serve as a high-level definition of the activities that take place during development. The well-known life cycles are, for example, the waterfall model, the prototyping model, incremental/iterative development, the spiral model [11]. There are also life cycle process models, which detail the activities of the life-cycle components. However, these activities are not ordered or structured within a sequence or time frame. They more or less constitute the building blocks of the software life cycle [12]. Life cycles are coarse representations of software activities, while software life-cycle processes are detailed descriptions of input/output and of the activities of life-cycle sub-process units. This information is structured in a standard format, like ISO/IEC 12207 and ISO/IEC 15504. This
is standard information, which could serve as a very appropriate reference for a course on software process.

Software process could also be defined as a product, which can be customized by users. This is the approach embodied in the Rational Unified Process (RUP) [13]. RUP is a Web-enabled software engineering process, which acts as an e-coach by providing prescriptive guidelines and templates. This process is designed to fit most organizational needs, and it must be adapted or configured for the appropriate environment, which means that it is a framework from which a specific process can be derived.

A course dedicated to software process must go further than a description of key practices, assessment approaches, the various life cycles and the many methodologies (small m) associated with software development. The purpose of the course is to enable students to understand the concepts behind software process, to increase their skills as team members and to contribute, eventually, to the improvement of their own software process environments. The main goal is to teach students what constitutes a software process activity, how and why it is related to other process activities, and what their outcomes are. The main course objective is to teach the students what they are doing while they build software, rather than listing the types of activities that can be carried out.

Students have one semester in which they understand the full scope of software process. In industrial environments, we observe that software engineers are often involved in only a few activities of a software process at a time. The learning curves of students in software process activities are steeper than those of professionals.

2: UPEDU

UPEDU (joyfully pronounced Yoopeedoo) is an acronym for Unified Process for EDUcation. The purpose of this project, which was undertook at the École Polytechnique de Montréal, is to design a software engineering process containing the main process activities or the activities that have a significant cognitive content. The objective is to introduce students to just the right number of software process activities to enable them to understand the role of software process in software development projects and to have them set aside activities which are more specific to a given task. Most students have little industrial software development experience, and many of the process activities have little meaning for them. The students will better understand these more specific activities later, when they are working in the ‘real’ world. The difficulty is to maintain a good balance in the process activities in terms of the various conceptual viewpoints they represent. Too many activities could reduce the learning process to a boring experience, however there should be enough activities to build a software process, which is of academic interest. Students are aware that the objective of the course is to learn the principles of software process and not to learn that specific process for its own sake, as it could never be used in an industrial environment.

2.1: The Physical structure of the UPEDU

In practice, the UPEDU is meant to be a teaching aid, aiming at students in software engineering in their 3rd or 4th year in a North American curriculum. The UPEDU consists of several artifacts:

a) A text-book, destined to the student, which contains 13 lessons of software process; we will describe them below

b) A CD-ROM, which contains supporting material:
   - The UPEDU process itself, in a browsable format.
2.2: The process model underlying the UPEDU

UPEDU is an academic customization of the RUP 2000, and we briefly outline here the major concepts that where used to create it. Like the RUP from which it is derived, the UPEDU is based on the concept that a software process is a collaboration between abstract active entities called roles which perform operations called activities on concrete, tangible entities called artifacts.

Figure 1 depicts this fundamental conceptual model using the UML notation for a class. Here, a role denotes one of several roles which may be played by an individual (or a small group of individuals) in the process. In the process, roles perform activities. An activity is a piece of work executed by one role. The granularity of an activity is usually a few person-days. An artifact is any piece of information or physical entity produced or used by the activities of the software engineering process. Examples of artifacts include models, plans, code, executables, documents, databases, and so on.

Figure 1 Conceptual model for roles, artifacts and activities

Figure 2 shows the graphical notation used to describe the various process components. A Process Component is a coherent aggregate of the Process Definition Elements organized from a given vantage point, such as a discipline (testing, for example) or the production of some specific artifact (requirements management, for example). The Designer role and the Reviewer role are different, and Architecting and Detailed Design are composite work items. A given role (see, for example, Designer) can work on different work items, artifacts or activities.
Figure 2 Graphical notation for process description

Figure 3 presents the workflows and the corresponding main artifacts that have been used to define UPEDU. Five engineering workflows and two management workflows have been retained and defined. Business Modeling, Deployment and Environment are workflows of the RUP process, which have not been kept for UPEDU. The UPEDU workflows are greatly simplified relative to the original RUP workflows. For example, only 37% of the artifacts are used in UPEDU, and only 10% of the guidelines. The emphasis of UPEDU is on providing a software process for learning, which will lead to a good understanding of the software process.
in one academic semester. We believe that too much material will hide the true nature of the process activities. To be efficient, a learning approach has to be focused on an understanding of basic knowledge and it has to be assumed that the students will bridge the gap when they work on a fully-fledged process during their careers.

UPEDU defines the skeleton of the process, which comprises workflows, roles, activities and artifacts, and their relationships. This is implemented in over 1,500 linked html pages. Students need to know what to do and how to carry out a given documented process activity. This precept defines the content of the course.

3: Case Study

UPEDU has been tested with a group of 30 graduating students in a project-oriented course. The students were organized in teams of six, some of which jelled, and spent an entire semester building a software system, from requirements to acceptance tests. A jelled team is a group of people that are so tightly knit, that the whole is greater than the sum of the parts. In other words, the production of a jelled team is greater than an unjelled team. Moreover, the enjoyment that team members derive from their work is greater then should be expected, given the nature of the work. These were accomplished in two iterations at the construction stage. The product was designed in UML and implemented in JAVA, and is a DECISION HELPER. It is a program, which computes the weight of various parameters in a decisional process, and is based on the Saaty algorithm [14]. The project involves the design of an interface for inputting the various parameters of the decision problem and the relative weights of the various parameters, the computation of the decisional matrix and the presentation of the results.

<table>
<thead>
<tr>
<th>Workflow</th>
<th>Artifacts</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirements</td>
<td>Glossary</td>
</tr>
<tr>
<td>Vision</td>
<td></td>
</tr>
<tr>
<td>Supplementary Specifications</td>
<td></td>
</tr>
<tr>
<td>Use-Case Model</td>
<td></td>
</tr>
<tr>
<td>Use Case</td>
<td></td>
</tr>
<tr>
<td>Actor</td>
<td></td>
</tr>
<tr>
<td>Boundary Class</td>
<td></td>
</tr>
<tr>
<td>User-Interface Prototype</td>
<td></td>
</tr>
<tr>
<td>Analysis &amp; Design</td>
<td>Design Class</td>
</tr>
<tr>
<td></td>
<td>Design Package</td>
</tr>
<tr>
<td></td>
<td>Use-Case Realization</td>
</tr>
<tr>
<td>Implementation</td>
<td>Component</td>
</tr>
<tr>
<td></td>
<td>Integration Build Plan</td>
</tr>
<tr>
<td></td>
<td>Release Notes</td>
</tr>
<tr>
<td>Test</td>
<td>Test-Case Procedure</td>
</tr>
<tr>
<td></td>
<td>Defect</td>
</tr>
<tr>
<td></td>
<td>Test Plan</td>
</tr>
<tr>
<td>Project Management</td>
<td>Project Plan</td>
</tr>
<tr>
<td></td>
<td>Measurement Plan</td>
</tr>
</tbody>
</table>

Table 1 List of the artifacts derived from the case study
All five teams were competing on the same project. We picked the best project realization and made it a case study for this course. The case study provides a sample of each artifact realized using the UPEDU software process. Students learning software process can then have a look at each of the artifacts resulting from the various activities. This seems to be an efficient learning tool, since students see the real end-products. Table 1 lists the 19 artifacts derived from the case study. All the artifact content is described in UPEDU and the templates are provided. These artifacts could also be used as examples for an in-house project.

4: Course Subjects

The structure of the content of the course is based on the UPEDU skeleton, but it is more than just a description of the components composing each workflow. The purpose is to present the cognitive activities that are involved in the process of creating software. These cognitive activities are framed by process structures, but they are likely to be similar, given another process as the basis for the framework.

The course is divided into 13 weekly subjects, which are presented in Table 2. The introductory subject, which consists of a short story about a lonely software developer facing the challenge of most of the process activities, is followed by Methods and Tools, which is a review of the main concepts used in UML. Software Life cycle and Software Process defines these two concepts clearly. The notion of the iteration is introduced. The five engineering workflows and the two management workflows are each presented in one week of lectures. Complementary subjects are: building a Software Team, Software Process Assessment and Measurement, Software process product, norms, standards and tools, and The future of software process (which is, in fact, an open subject). The following describes the cognitive objectives of some of these subjects.

The purpose of the short story is to show that the basic process of developing software is independent of the size of the organization. This lonely developer will face the same cognitive and organizational challenges that big organizations are likely to face. As well, the story serves to illustrate what is really basic in software process, that, unlike a cooking recipe, it is more than a list of activities. It also introduces the need for iteration, which is a key concept and another way in which software process differs from the recipe.

Methods and Tools focuses essentially on a review of the Unified Modeling Language, which is the notation used to illustrate the various artifacts. This topic provides completeness and serves for review purposes only, since students must be familiar with this notation before getting into the subject of software process itself. Indeed, an important intellectual challenge might be to learn object-oriented methods and software process in a single semester.

There is some confusion in the literature between software life cycle and software process, as illustrated by the IEEE 1074 Standard for developing Software Life-Cycle Processes. Software products have life-cycle phases through which the product goes, while software processes have workflows, which define activities, which take place during the various phases. The concept of iteration illustrates the merging of the two concepts of process and life cycle.

The five engineering workflows are presented in terms of the intellectual activities needed to realize them. For example, there is a presentation on the differences between the cognitive activities associated with analysis and design. Students realize that analysis and design are not sequential activities as in the waterfall model, but rather opportunistic activities carried out at the most appropriate time by software engineers. Activities in process workflows are presented in terms of cognitive and intellectual challenges, which makes them different one from another. Students realize that creating software is nothing like following a recipe. They
know the names of the major activities, but the challenge is combining them. It is the efficiency of the mixing process that makes for a product of great quality.

<table>
<thead>
<tr>
<th>A Short Story about Software Process</th>
</tr>
</thead>
<tbody>
<tr>
<td>Methods and Tools</td>
</tr>
<tr>
<td>Software Life Cycle and Software Process</td>
</tr>
<tr>
<td>Requirements Workflow</td>
</tr>
<tr>
<td>Analysis and Design Workflow</td>
</tr>
<tr>
<td>Implementation Workflow</td>
</tr>
<tr>
<td>Tests Workflow</td>
</tr>
<tr>
<td>Software Team</td>
</tr>
<tr>
<td>Software Project Management Workflow</td>
</tr>
<tr>
<td>Configuration and Change Management Workflow</td>
</tr>
<tr>
<td>Software Process Assessment</td>
</tr>
<tr>
<td>Software Process Measurement</td>
</tr>
<tr>
<td>Software Process Metamodel</td>
</tr>
</tbody>
</table>

**Table 2 List of subjects**

*Software Team* is a great topic for students. They learn all the intricacies of the jelled-team approach. They realize that team activities can be either superficial or very engaging. They come to understand the difference between personal involvement in a task and an assigned work schedule. They see the benefits and also the limitations of an e-team. The jelled team is not for the Internet fan yet.

Software process measurement opens up a new field in software engineering. Three measurement approaches are presented. The certification-like assessments based on the CMM are presented. Process workflows can also be evaluated based on effort measurements [15, 16], cognitive activities can be measured to enable a better understanding of the activities that compose the workflow [17, 18]. These measurement approaches emphasize the roles of the process in software development. Processes define the organization of work, which can be certified; provide a structure for the identified workflow, which can be managed by appropriate resource measurements; and, finally, provide prescribed tasks which result in cognitive activities.

5: **Conclusion**

This course on software process was given for the first time in the winter term of 2000 to 15 volunteer students. The Software Process course will be mandatory in our new software engineering program starting in the fall term of 2000. All the material, which includes 20 transparencies per chapter, is made available to the students through the course Web site. Every week there are quizzes, for which students need to browse through the UPEDU Web site or study the various artifacts to find the correct answers. A formal evaluation of the course by our pedagogical services gave it an outstanding rating.

6: **Acknowledgement**

The UPEDU Project has been sponsored in part by Rational Software Corp. We are grateful to Houcine Skalli form S2I inc. for his work on the Case-Study. Thanks to John Slavich for his throughout revision of the content.
7: References