Abstract

The Information Communication and Technology – ICT industry is facing a market of constant changes and challenges. These characteristics demand the ICT professionals to have a wide vision of the problem, rather than just knowledge on the technology. In this context, the objective of this article is to propose an innovative pedagogical methodology based on PBL (Problem Based Learning) to improve the learning effectiveness in software engineering, through the implementation of software factories, where students can work together to solve real problems. This methodology is running in a graduate course in Software Engineering managed by CESAR, a research institute with experience in the development of innovative software. Some important results about this course are presented and discussed in this paper.

1. Introduction

The Information Communication and Technology – ICT industry is facing a market of constant changes and challenges. Professionals in this area frequently need to adapt their concepts, methods and techniques to specific situations of the market problems, in general, influenced by variations of scope, costs, timelines and risks of software projects. Moreover, the evolution of the ICT market, as a competitive advantage of the organizations, has been fostering a strong demand for different and innovative ways to develop software with more quality, flexibility and productivity returns. These characteristics of the ICT market enact a demand the professionals, particularly software engineers, not only knowledge about technologies, but also a wide vision of the problem being solved, business understanding, entrepreneurship background and interpersonal skills, all related to the practical experience of the work. In this context, this article proposes a pedagogical methodology based on PBL (Problem Based Learning) [1] to Software Engineering education and presents some important results obtained from its application into a graduate course in Software Engineering with professional emphasis. PBL is used in this methodology to improve the effectiveness of learning, promoting the ability of the students to work in teams to solve problems and also encourages the development of their skills and attitudes, including teamwork and self-directed learning skills, cooperation, ethics and respect for other people’s points of view [2,3].

In [4], Peterson appoints three important criteria that promote optimal learning in PBL approach:

1. The approach provides an environment where the student is immersed in practice;
2. The student receives guidance and support from his/her friends and peers; and
3. The learning is functional, based on solving a real problem.

Having as reference the criteria defined by Peterson, the methodology provides an environment based on Software Factory in which the students are immersed in practical software development projects taken from real clients, supported by processes, roles and metrics to control the results achieved.

The adoption of PBL is not an easy task and such method is frequently a surprise to students and educators, due to its paradigm totally different from conventional teaching, based on teacher-to-student unilateral learning. In [5], authors describe their experiences with a senior undergraduate course in Management Information Systems (MIS) Design. In the early stages of the factories, some problems were detected, in particular related to the requirements phase, in which students usually take a superficial view

---

1 C.E.S.A.R. – Recife Center of Advanced Studies and Systems
http://www.cesar.org.br
of the problem and focus on potential solutions rather than understanding the many implications of the problem requirements. In this context, the methodology proposed here is recommend for graduate students, considering their experiences with software development (in general, 1-2 years) and professional maturity levels in collaborative work.

This paper is organized in four sections. Details about the methodology are presented in Section 2 and the application of this methodology together with the results achieved in two graduate classes are discussed in Section 3. Section 4 presents some concluding remarks and future works.

2. The Methodology

Similarly to other engineering areas, Software Engineering is concerned about the intersection of people, technology, domain, and opportunity aspects. In this context, the methodology proposes the use of software factory as the appropriate environment to implement the practice for software engineering students, according to criteria (1) of Peterson, as mentioned in the Section 1.

### 2.1. Human capital

In the human capital context, the methodology is supported by four roles that interact with each other to run the educational methodology along the semester. The term “software factory” is erroneously associated to the fact that software development is a kind of mass-production of industrial products. Despite of this, the concept of a factory also implies a particular way of organizing work with considerable job specialization, formalization of behavior and standardization of work processes [6]. The software factory model used by the proposed methodology aims at the standardization of good practices, in order to gradually improve tools and techniques and to establish appropriate control through metrics definition and evaluation [7].

To support this lifecycle, some content, guides and processes are necessary. Also, stakeholders and their roles need to be identified, in order to map their participation and influences in each step. These elements will be discussed in the following subsections, and the implementation of this lifecycle is discussed in Section 3.

#### Table 1. The software factory lifecycle.

<table>
<thead>
<tr>
<th>HUMAN CAPITAL</th>
<th>SWF LIFECYCLE</th>
<th>GUIDANCE &amp; SUPPORT</th>
</tr>
</thead>
<tbody>
<tr>
<td>SWF Team SWF Monitors</td>
<td>SWF Creation</td>
<td>SWF Building Process</td>
</tr>
<tr>
<td>SFW Team SFW Monitors</td>
<td>SWF Execution</td>
<td>SWEBOK Content Real projects (RFP)</td>
</tr>
<tr>
<td>SFW Team SFW Monitors Tutors Clients</td>
<td>SWF Control</td>
<td>Control Process</td>
</tr>
<tr>
<td>SWF Team SFW Monitors Clients</td>
<td>SWF Improvement</td>
<td>Control Process</td>
</tr>
</tbody>
</table>

The term “software factory” is erroneously associated to the fact that software development is a kind of mass-production of industrial products. Despite of this, the concept of a factory also implies a particular way of organizing work with considerable job specialization, formalization of behavior and standardization of work processes [6]. The software factory model used by the proposed methodology aims at the standardization of good practices, in order to gradually improve tools and techniques and to establish appropriate control through metrics definition and evaluation [7].

To support this lifecycle, some content, guides and processes are necessary. Also, stakeholders and their roles need to be identified, in order to map their participation and influences in each step. These elements will be discussed in the following subsections, and the implementation of this lifecycle is discussed in Section 3.

#### 2.1. Human capital

In the human capital context, the methodology is supported by four roles that interact with each other to run the educational methodology along the semester. The first role is the software factory team (SWF Team), that are designed to exploit different capabilities in software development. Such team is composed by students, who play all roles in software development such as project manager, requirement analyst, software architect, software, quality and test engineer. The methodology recommends teams with 5 to 7 members, therefore each student performs more than one role in the team. It’s important to emphasize that all students are involved in each one of the steps of the software development process.

The second role of the methodology is the Tutor, who is responsible to teach the planned program for each discipline, coordinate academic activities, and act like a consultant in the specified discipline context. Besides that, they systematically evaluate students, and provide the Software Factory Monitors information about the practical activities required during the disciplines. Therefore, besides academic and research experiences, it is essential that the tutor have enough market skill in the knowledge area of his/her classes, as well as successful and relevant participation in software development projects.

The Software Factory Monitor (SWF Monitor) is responsible to track the development of software factories, and to conduct the tracking process. This role is also responsible to generate reports about software factories results and status and to evaluate the performance and commitment of each team member.

The Client, just like in real market environment, is the one who demands projects to the software factories through a Request for Proposal – RFP. This document has a brief explanation on the client’s necessities and he must be available to interact and track each software factory according to the project development. The
client is also responsible to answer the “client satisfaction evaluation”, which is part of the assessment method applied to the factories.

2.2. Guidance and Support

As guidance and support, the methodology includes the processes to build and to track a software factory, a study guide providing content to support disciplines and a RFP (Request for Proposal) model to request real projects.

2.2.1 Software Building Process

A process is a set partially ordered activities (or steps) to reach an objective. In fact, process lifecycles are defined by some reasons, including to support improvement of the process and to support process management. In this context, the pedagogical approach proposed here includes a process to build the software factories as part of the methodology [7]. It’s organized in seven steps:

1. **Forming software development teams**: the division of class into teams is executed by considering aspects like affinities skills and individual abilities of each student. This is done through a questionnaire of profile’s analysis and interviews, in order to avoid concentration and/or insufficiency of abilities in one team. Additionally, MBTI method [10] may be applied to identify characteristics and preferences of each student.

2. **Factory models study and appropriation**: students understand and analyze the most relevant software development processes, which helps choosing and formally specifying one to be applied in their software factories.

3. **Software factory process definition**: this step includes the detailing of the development process, activities flow, team roles, artifacts and templates.

4. **Define software factory infrastructure**: in this step, students need to define the necessary infrastructure for the factories functioning, such as: the website, repository of templates and artifacts, SLA (Service Level Agreement) contracts, software development metrics, software development environment and the quality assurance of products and processes.

5. **Define software factory business model**: this step includes the definition of software license, generation of copyright policies for the products, creation of the communication model with the clients, production of the contract models and templates of commercial and technical proposals.

6. **Define internal evaluation policies**: this stage contemplates the evaluation of processes and human capital as executor of this process. Additionally, the contract and renewal qualification policy for the human capital of the software factory are defined.

7. **Client evaluation policies**: in this step each software factory will need to define the client satisfaction criteria and strategies to improve the assessed products and processes.

This process is running since 2003 and has been efficient as a mechanism of organization of teams, definition of a software development process and the project management associates to these teams [3].

The following subsections comment each steps of this process.

It is important to emphasize that, for each new project it is necessary a new website, providing a proper identity for each project, with the objective to control the software development activities by the team members and other stakeholders. To increase the agility of construction of the sites, the use Content Management System (CMS) is recommend, therefore simplifies and saves time in the creation and maintenance of the information.

2.2.2 Content

According to IEEE Standard Glossary of Software Engineering Terminology, software engineering is defined as: (1) The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; (2) The study of approaches as in (1).

Considering the objective of “to promote a consistent view of software engineering worldwide” and “to characterize the contents of the software engineering discipline”, the methodology recommends a curriculum based on SWEBOK (Software Engineering Body of Knowledge) [9]. Within this context, the curriculum is organized in seven SWEBOK-based disciplines, and each one is divided into three modules: basic, intermediate, and professional. These disciplines are structured with a content to support the resolution of software factories’ problems, and their modules are interleaved to promote an iterative and incremental development. The disciplines proposed by the methodology are: (1) Software Factories; (2) Application and User Interface Requirements; (3) Project Management; (4) Architecture Oriented Development; (5) Reuse
Engineer; (6) Technology of System Interoperability; (7) Verification and Validation of Systems.

The overall program was designed with weekly modules of each disciplined, where there are 15 hours of classes, 12 hours of guided practices and an individual evaluation by the end of the week. The hours of practices are used to help students implement the theory, presented during classes, on their real problem faced by the software factories. There are tutors to give support on the execution of these activities.

### 2.2.3 Real Projects

The software project must be a demand from a real client. Therefore, the methodology defines some criteria to support the definition of such projects. First, problems must have relevant concepts and principles to the content domain. Thus, the process begins with the identification of primary concepts or principles that a student must learn. Second, problems must be real, with concerns and issues. Each project is submitted by the clients and they must be compliant to a set of requirements, related to the topics: innovation, relevance to industry, business model and development process.

The clients need to motivate students and engage them into the problem. It is important to emphasize that, if students are engaged in an authentic problem, then they must own the problem. Learners perceive problem as a real problem with great relevance [1].

### 2.2.4 SWF Control Process

The SWF control process is based on the market reference model of maturity CMMI (Capability Maturity Model Integration) [11] that gathers best practices from the industry which can be implemented in software projects. The objectives of this process are:

- To provide a diagnostic tool to ensure students are progressing adequately towards achieving the desired learning goals;
- To provide a multidisciplinary approach to collectively assess students;
- To set up an assessment process as part of the learning practice;
- To create an assessment process that can be repeated over time; independent on the context. In this process, given the same variables, they should produce equivalent results; and
- To provide students a feedback of their evolution in a fair and objective manner.

Having these objectives, the control of the software factories are made through the application of following evaluations:

- **Independent Software Factory Evaluation**: the factory monitor collects all the evidences and information from the factory and the project being developed with the objective of analyzing the progress and continuous execution of the activities.
- **Alignment Evaluation**: this activity has the objective of analyzing whether or not the members of the same factory, are aligned with the objectives and activities being developed.
- **Client Satisfaction Evaluation**: this is the evaluation of the client satisfaction level related to the service delivered to him.
- **Status Report Evaluation**: this is the moment when the team realizes a formal presentation of the project and the process being executed.

Within this scope, the tracking of the software factories are planned by the Software Factory Monitor, and such plan is maintained and updated with the support of Moodle [12], which is a free open source course management systems, designed to support educators maintain effective online learning communities.

Besides the plan, other artifacts support the tracking process, such as:

- **Work Product Checklist**: this artifact list the criteria to objectively evaluate the work-products elaborated by the software factories, in terms of what should be investigated in order that the artifact produced by the software factory can by compliant to the requirements seen on the discipline of the course and requested by the client.
- **Process Checklist**: this artifact is similar to the work product checklist so that the processes executed by the software factories, can be objectively evaluated by the software factory monitor. This checklist contemplates the requirements of the process learned on each discipline and supports the monitor finding whether or not the defined process is compliant to the executed process.
- **Tracking Report**: this artifact contemplates the results of each evaluation executed by the software factory monitor. It presents the findings of each kind of evaluation, in terms of the non-compliances and best practices executed by the teams.
• Alignment Evaluation Questionnaire: this artifact is a subjective questionnaire that focused on processes practices executed by the team.
• Client Satisfaction Questionnaire: this artifact is an objective questionnaire that contemplates questions regarding the interaction of the software factory team with the client.

It is important to emphasize that the methodology includes individual and collective evaluation. The first one is focused on individual evaluations about concepts and fundamentals learned from the disciplines content and it is conducted by the tutor. This evaluation is not discussed in this article (more details in [13]). The last one is focused on obtaining the overall result of each software factory and it is conducted by the SWF monitor.

Finally, the market reference model of maturity CMMI was chosen as a model that gathers best practices from the industry that can be implemented in software projects. Specifically in the context of tracking software factories, as part of the evaluation process, the Process and Product Quality Assurance- PPQA – process area is the one that best contemplates a guide for the evaluation, in order to verify if the defined process, of the software factories, are being executed according to what was planned [14].

3. Case Study: A Master Course in Software Engineering

The methodology proposed here, is being applied in a professional Master Course in Software Engineering (MCSE) since August 2007, a kind of master course with focus on market demands. The main objective of this course is to form software engineers with a wide vision of software lifecycle and all resources involved (People, Processes and Infrastructure). Additionally, the course helps understanding how to apply the acquired knowledge to solve real and complex problems using innovative ways. The course takes one year: the first six months are focused on the disciplines’ classes and the works in the software factories. The following six months are dedicated to the Research and Development – R&D project, with a report based on individual contributions of each student.

The first class of MCSE was composed of 19 students, with average age of 27 years and they were professional whose abilities were in the most of the cases: test engineer and quality engineer. This assessment happened from August 2007 to February 2008, during the first stage of the master’s program. In this class, four software factories were created. The second class of MCSE was smaller than the first one, it was composed of 10 students, with an average age of 30 years and their abilities are mainly software engineer and project manager. The second group had a higher professional maturity level than the first, distributed in two software factories. This assessment happened from March 2008 to September 2008.

3.1 SWF Creation and Execution

The process of building software factory was planned to start with the first discipline of the course, the basic module of Software Factory. In this module, students are guided to form their groups and to structure their teams. The activities of this process are presented in the Section 2.2.1. In the first class of MCSE, the process of building software factory started with initially planned. But, this strategy showed that this step demanded much time and work that conflict with the objectives of the discipline, focused on software factory models and software development process definitions. To minimize these problems, in the second class, this step was anticipated in two weeks before the first discipline, by the creation of one more step in the process called “software factory building practice”. So, in the initial stages of the basic module of Software Factory discipline, the software factory team was already defined and ready to work on models and processes, the goals of this discipline.

Along the execution of software factories, there were some difficulties, in particular in the first class of MCSE. Some of these difficulties are related to guarantee an incremental software development which contains a complete software lifecycles, including the later codification step. These characteristics of software development process based on an extensive and descriptive production of artifacts, concentrated in the early stages, such as project plan, requirements documents, project design, and so on, caused a certain apprehension to the students who need to see a system functioning on the computers. To minimize this problem, the sequence of disciplines modules was modified, including discipline modules more related with codification step, such as Architecture Oriented Development and Reuse Engineer, to the first groups of disciplines. Moreover, the methodology defined a more detailed evaluation calendar, with four previously planned status reports, one for each interaction of the software development cycle.
Finally, different from the first class, the second really included the Client in the software development process. Beyond the specification of the project through a RFP, each software factory maintained frequent meetings with the clients to validate requirements, interfaces and changes in the projects.

### 3.2 SWF Control and Improvement

The control process on each software factory is executed according to the steps described in the Section 2.2.4. In the first class of MCSE, a lighter process composed of five evaluations (3 Status Reports, 2 Independent and 1 Alignment), in which the first one began 40 days after the first discipline, showed a certain instability in the software factories performances, although always in a crescent curve, as illustrated by the Figure 1. On the other side, the class profile with less experience in the overall software development process implied in a great evolution along of this practice. As related in [12], by observing the graphic in the Figure 1, it is possible to see that all software factories presented a final grade, on the last status report, greater than the first status report grade. To emphasize, the biggest progress observed is from software factory 3, where the difference from the first grade and the last grade is 3.67 points, representing a progress of 73%.

For the second class of the MCSE, there was a control process more complete than the first, with eleven grades obtained by different evaluations (4 Status Reports, 5 Independent evaluations, 1 Client Satisfaction evaluation and 1 Alignment evaluation), as illustrated in Figure 2.

Note that, the inclusion of the Client Satisfaction evaluation, which brings an external evaluation and the direct participation of the market, an essential element to the effectiveness of this methodology. The graph presented in the Figure 2 shows the timeline evolution of the students’ performances. In both software factories, it is possible to see the increasing of the scores. This characterizes the students’ evolution during the Master’s course. Most of the grades were greater than 7 (the minimum grade expected), and they evolved to greater values along the time.

Special attention to the second factory in Figure 2, in which the grade evolution is not linear. There is a high decrease in the degrees of the independent and status report evaluations. This happened because the project’s scope was completely redefined one day before the status report meeting, and this situation caused a delay due to the re-planning and refactoring of the software artifacts produced so far, with a negative impact over the grades. This situation reflects common negative impacts from not planning changes.

### 4. Conclusions and Final Considerations

This paper proposed a methodology based on PBL, implemented through software factories, to Software Engineering Education and Training. As the main contributions, the methodology elements – human capital and processes – were indentified along the software factory lifecycles. This methodology has some key components: a process to build the software factory, a process to track the software factory works and a specific role, the software factory monitor, who is responsible to guarantee the quality of the process application and conduction.

This methodology still has some improvements to be accomplished, but it has already presented significant success.

In the first class of MCSE, some critical issues were identified and managed: (1) the unfamiliarity of the tutors with the methodology and PBL background, (2) the selection and conduction of real world projects with real clients, (3) the alignment between content of discipline and its application in the SWF projects and,
(4) the appropriate adjustment of students to a fast learning approach executed on strict six months.

We have some difficulty to change the tutors’ behavior in order to perform a coach and consultant role. The tutor must enforce the students’ thinking by telling them what to do or how to think, rather than tutoring, they should support the students along the projects development, also becoming responsible for the produced software. To achieve this, a tutor training must be considered as a key factor for the success of the proposed methodology.

With respect the real projects, learners must have ownership of the learning or problem solving process as well as having ownership of the problem itself. However, we observed that it was necessary to specify a pre-defined group of projects with a considerable grade of complexity and market characteristics. In the first MCSE class, for example, one of the selected projects presented very simple requirements to exercise the complete software practices, and in the second class, one project had few codification activities and thus, was aborted on the second month of the course.

The software development process in the software factories may be supported by the contents of the disciplines. These disciplines may provide the students with the knowledge needed in solving the problems faced by the factories. To achieve this, both disciplines and project phases may be synchronized. For example, during the project management discipline, the software factory is supposed to be working in the project management plan. It is important to call attention to two points. First, the manner of students use information resources and instructional materials as sources of information. The materials do not teach, but rather support the student inquiry or performance. Second, the development of real world systems are iterative and incremental, and the knowledge gained into a discipline may be applied in later stages of software development.

In the beginning of the course, students present some resistance to the course rhythm. Therefore, in the end of the second month, they were suited to the rhythm, guided by the project velocity. We extracted from the satisfaction students points of view, the evaluation of the course. The results showed 100% of affirmative responses to the two questions: (i) “Does the MCSE provide improvements in your professional performance?” and (ii) “Would you advise someone to study in MCSE?” Besides, there were positive testimonies such as:

“The MCSE is a real professional course. It is not an uncovered academic course. It is a legitimate professional master course, with an experienced and specialist group of tutors, in all of the covered areas. The MCSE focus in solving complex problems extracted from real world, and the produced solutions are applied and possible to be used. There is also a different methodology simulating a software factory and students are inserted in a safe environment with a number of day-by-day conflicts, problems, activities and solutions. G.A.”

It’s important to emphasize that the period of six months for a project development is totally aligned with usual time-to-market of the information systems development.

The second class was focused in the SWF building improvements, in making the clients more operating and participative, in establishing a strong control of the software factories by the SWF monitor and in collecting metrics to new improvements.

The client is the most important and critical element to represents the reality of the market, thus, his/her participation and engaging in the project is fundamental. Besides to specify the demands and macro requirements, it is very important that the client participate in the software factory evaluations, status report meeting, verifying and validating deliverables according with pre-defined plans.

The definition of more restrict processes provided a higher stability in the tracking and evaluation of the software factories. In this context, we are planning as future work, the definition of a metric model to support the tracking process, so that the conclusions can be quantitatively evaluated upon the possession of metrics from software factories. Following the same considerations of PPQA process area, in this context the Measurement and Analysis process area could also be used to support the definition of such process. Nevertheless, it is also an improvement gap the deployment of a tool to support software factories tracking, in conjunction with the learning environment used by the master course. In the same line, of tools to support the process, another improvement identified, in order to best support the process of tracking the resolution of the issues found, during status report or independent evaluation, it is seen the necessity to deploy a bug tracking tool, which could support the process of tracking non-compliances that need to be resolved. As results, we can state that the students, immersed in the learning context, are gathering great practical knowledge and support their introduction into the market, that is in lack of qualified professionals.

Finally, as usual practice, the results of each classes of the MCSE are compared among them. In the comparative analysis between the classes 1 and 2, we
observed a positive convergence of the software factories performances, as illustrated in the graphic on the Figures 1 and 2.

Nowadays, a third class is in course, with the focus to explore the innovation characteristics of the projects, essential elements to learning concepts in Software Engineering. To achieve this, we defined and research specific area focusing in interactive applications for Digital TV. This is a practice to allow the exploration of innovative characteristics of the application, with a greater focus on the product besides on the process.

5. References