Towards a Model II Theory-in-use for young software engineers and small sofware teams

Vincent Ribaud, Philippe Saliou

Towards a Model II Theory-in-use for young software engineers and small sofware teams

Vincent Ribaud, Philippe Saliou
Département d'informatique (Computer Science)
Université de Bretagne Occidentale
Brest, France
{Vincent.Ribaud, Philippe.Saliou}@univ-brest.fr

***Abstract*— Small teams have to transform in a learning organization to cope with the changes in IT. Argyris and Schön distinguish single-loop and double-loop learning [9]. Single loop learning happens when unintended or counterproductive consequences lead to a change in action but not in the governing variables. Another possibility is to change the governing variables themselves and is called double-loop learning. Single-loop learning is induced from Model I, a prevalent model of theories-in-use - those that can be inferred from action -. Argyris and Schön look to move people from a Model I to a Model II that fosters double-loop learning. In the software engineering field - and especially in small teams, developing a reflective thinking and enhanced learning is a vital issue. We intended to develop these issues in the course of a Master program in Information Technology and Software Engineering. The last year of this program is performed under 'sandwich' conditions with an alternation of study periods in university and training periods in industry. Moreover, alternated university periods are dedicated to a long-term team software project. The education system is a reflective practicum. Such a practicum provides students, working in groups, with the possibility to reflect on her/his action and that may help making explicit theories-in-use. Several reflective practices are seamed in the course of the project providing an students with education of reflective thinking. The work placement system introduces a new challenge that is to relate the university and industrial phases of the student's experience. We propose to use journal writing as a tool to record young engineers' behavior and to extract meaning from events and experiences. The first goal of these different practices is to sustain a reflective thought that may help to question espoused theories and to reveal theories-in-use; a more ambitious goal is that the whole team acts as a learning organization with a theory-in-use mastered by Model II. We report on an experimental case study using a project journal supported by semantic wikis.**

Keywords-component; reflective practitioner, software engineering processes, organizational learning, journal writing.

I. INTRODUCTION

Argyris and Schön define learning as the detection and correction of error. "*Single-loop learning occurs when errors are corrected without altering the underlying governing values. [...] Double-loop learning occurs when errors are corrected by changing the governing values and then the actions* [1]". Small organizations – and especially small software project teams – need constantly to adapt their task force to products or services to be delivered. The notion of 'learning organization' is a central point in this challenge and is defined as an ideal "*towards which organizations have to evolve to be able to respond to the various pressures* [they are faced to]" [2]. This paper presents elements of a work placement Master program in Software Engineering focusing on aspects that may favor double-loop learning, individually as apprentices or collectively working in a small team.

In a work placement program, apprentices alternate study periods at the academy with work periods in the industry. A major challenge is to avoid that apprentices will 'climb two ladders simultaneously'. Our system organizes all academic apprenticeships inside a "learning by doing" environment, very similar to software development studios [3], but embracing, as far as possible, most software lifecycle processes defined in the ISO/IEC 12207 standard [4].

Schön's reflective practitioner perspective [5, 6] guides professional creative people to reflect about their professional creations during (*reflection-in-action*) and after (*reflection-on-action*) the accomplishment of the creation process. To educate the reflective practitioner, Schön recommended looking at traditions of education for artistry – and especially the architecture studio. "*Studios are typically organized around manageable projects […]. They have evolved their own rituals, such as master demonstration, design review, desk crits, and design juries, all attached to a core process of learning by doing*" [3, p. 43]. Schön qualified as a reflective practicum this learning environment. This analogy was used to provide a suitable educational environment for software design by Tomayko at Carnegie-Mellon [3], Kuhn at MIT [7] or Laplante [8] at PSU.

We designed and implemented, in 2002, an education system called "Software Engineering by Immersion". The system is entirely based on performing complete development cycles of a software project and accomplished in two iterations followed with an operational internship in a firm. Students work in teams to analyze, design, implement, test and document a software project relying on strong software engineering principles. No regular software engineering courses are delivered. During the first iteration (4½ months), students are swapped around among the different tasks required by engineering activities, with strong guidance from the coach. During the second iteration (2 months), teams are autonomous in completion of the project. A process reference model simplified from the ISO/IEC

12207 provides a breakdown into process groups - processes - apprenticeship scenes. Issuing from professional didactics, the analysis of activity distinguishes two kinds of activities [10]: productive activity, which is transformation of the reality, and constructive activity, which is transformation of the subject through its own practice. Our approach tries to combine learning through environmental immersion - intended to offer to students a reflective practicum - and a well-structured learning process - that aims to organize productive and constructive activities. As realistic working situations were experienced, it provides students with progressive understanding of software engineering. In order to develop a reflective analysis, several reflective practices have been borrowed from different educational system and seamed into development processes, mainly the apprentice-coach dialogue, students peer review, activity tailoring, retro-engineering.

In 2007, local employers in Brest demanded for employees under 'sandwich' (or work placement) conditions and we adapted the "Software Engineering by Immersion" programme to run as a work placement course. During 12 months, the work placement student is a full-time employee, although also attending university for certain periods. Salary is about 80% of the salary corresponding to the job that the course leads to. We assigned the second iteration to the industrial periods and first iteration (at the university) and second iteration (in the industry) were intertwined with a two weeks / two weeks rhythm.

Nearly all young engineers engaged in our program are working in large companies with a structured corporate baseline. Despite of this fact, we observed last year that most of them worked either alone, or in very small teams. Thus, we found the situation of our young engineers very similar to what could happen in a Very Small Enterprise (VSE). Also, the reflective practicum where students are plunged during the university periods reproduces as far as possible a small software company with its proper equipments, tools, methods, culture and people. So, we decided to revisit our system in order to satisfy, as far as possible, challenges of learning and producing in a VSE. The key point is to not overload software engineers with practices they would feel useless but to seam reflective practices in the course of the project in order to make them natural and value-adding to engineers. Faced to the problems to relate the university and industrial phases of the student's experience, we choose to promote journal writing to enhance reflective practice.

The next section overviews Argyris and Schön theory of action, their application to software engineering and some related work. In section 3, we discuss about reflective practices. Observing the course of apprentices' projects is discussed in section 4. We conclude the paper with a discussion and perspectives.

II. ACTION THEORY AND SOFTWARE ENGINEERING

A. Software Engineering by Immersion

Authors defined an apprenticeship/production framework called ILI (Ingénierie du Logiciel par Immersion, Software Engineering by Immersion), based on a reference model, a development cycle, a WBS (Working Breakdown Structure: *deliverable-oriented hierarchical decomposition of the work to be executed by the project team to accomplish the project objectives and create the required deliverables* [11]).

The Process Reference Model (PRM) is adapted and simplified from the ISO/IEC 12207:1995 standard and its amendments [9]; we are using 3 *process groups* organizing 13 processes: *Software Project Management* (Project Management, Quality Assurance, Configuration Management); *Software Development Engineering* (Requirements capture, Software Requirements Analysis, Software Architectural Design , Software Detailed Design, Software Construction, Software integration; Software qualification testing); and *Software Development Support* (Infrastructure Management, Life Cycle Model Management, Documentation Management, Installation-Operation).

We use a Y-shaped life cycle that separates resolution of technical issues from resolution of feature issues [12].

The WBS has a structural and a temporal decomposition. Each process is structurally decomposed in exemplar Software Engineering activities (*SE activities* to distinguish it from the activities in the 12207 sense) that may have slightly variation from a project to another. A current issue is to align SE activities with Base Practices of the 15504 standard [13]. The level of detail of a PRM such the 12207 is not sufficient. Therefore it needs to be supported with a comprehensive set of base practices: "*an activity that, when consistently performed, contributes to achieving a specific process purpose* [13-1, p. 3].

The WBS is temporally organized in stages (in our case, 9 of 2 week each). The planning of each stage is divided in several work scenes that carry on SE activities. Scenes will be performed by team members and ought to produce artifacts. We introduce this year a project journal intended to record all elements of interest in the course of the project. Boud presents journal writing "*as a form of reflective practice, that is, as a device for working with events and experiences in order to extract meaning from them*" [14, p. 9]. The project journal uses semantic wikis in order to record information. Team members can record events as they happen but have to systematically fill the wiki at the end of each phase. Semantic wiki is the most flexible tool in order to record and shape a structured content. The journal records the "historical" context of resources' use and products' production. An unusual usage is the recording of products and documentary resources itself because they are main components of software engineers' activity as they describe the inputs and outputs of the activity. We provide the journal with a hierarchical structure. At the first level, each participant records her/his individual course of action in semantic wiki pages of the class *Performed Activity*. At a second level, a collective action is recorded in pages of the class *Course-of-action Unit;* it involves parts of several individual *Performed Activity* taking place synchronically or sequentially. At the higher level, complex and collective interactions are recorded in *Step-of-action* pages sequencing and embedding *Course-of-action Units*. These pages are created and updated at different rhythms: each 2-3 days for *Performed Activity*; each 2 weeks - at the end of a stage - for

Course-of-action Unit; three times per project for *Step-of-action*. The latter writing is more an activity analysis than diary writing and apprentices establish links with the PRM. A picture of these interlinked concerns is given in figure 1.

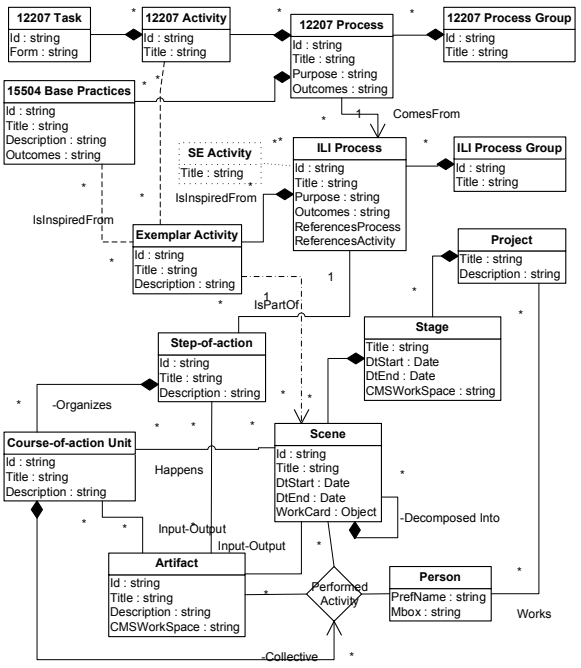

Figure 1. The upper half depicts reference models. Within process groups, processes are defined with purposes, outcomes, activities and tasks. The 15504 Process Assessment Model expands the 12207 Process Reference Model by including a set of base practices. ILI processes, drawn from 12207 processes, are performed through exemplar activities. The lower half represents the enacted project. The lifecycle of a project is organized into stages composed of scenes. During a scene, persons perform a SE activity inspired from an exemplar activity but contextual to the project. Input and output work product (artifact) are associated to the scene, the activity and the process. Self-observing the action leads to a rebuilding of project processes into steps of course-of-action units.

B. Argyris and Schön's Theory of Action

According to Argyris and Schön, people design and guide their behavior by the use of theories of action. "*Espoused theories of action are the theories that people report are governing their actions. Theories-in-use are the theories of action that actually govern their actions*" [15, p. 7]. Argyris and Schön argued that, if espoused theories vary widely, theories-in-use do not. They labeled the most prevalent theory-in-use Model I. "*Model I theories-in-use are theories of top-down, unilateral control of others for the actors to win, not to lose, and to control the environment in which they exist to be effective*" [15, p. 7]. They argued that with such a theory-in-use, problem solving works for issues that do not require that the underlying assumptions of Model I be questioned (*single-loop learning*). Model II theories-in-use make possible for people "*to have problem-solving skills that question the governing values of their theory-in-use*" [15, p.7] (*double-loop learning*).

Models of theory-in-use

Model I and Model II looked to three elements. *Governing variables* are values that actor seek to satisfy [1]. Each governing variable can be thought as a continuum with a preferred range (e.g. not too anxious, but not too indifferent) that people are trying to keep in these acceptable limits. *Actions strategies* are sequences of moves used by actors in particular situations to satisfy governing variables [1], there are the moves and plans used by people to keep the governing variables in the preferred range (e.g. to practice a physical exercise to eliminate stress). Consequences happen as results of action. Consequences can be intended – those that the actor believes will result from the action and will satisfy governing variables (e.g. to feel him/herself better after a sportive effort). Consequences can be unintended but they are designed because they depend on the theories-in-use of recipients as well as those of actors.

Single and double-loop learning

When the consequences of an action strategy are as the actor wanted, then the theory-in-use of that person is confirmed. If there is a mismatch between intention and outcomes, consequences are unintended. Argyris defines learning as the detection and correction of error. The first response to error is to search another action strategy. "*Single-loop learning occurs when errors are corrected without altering the underlying governing variables*" [2, p. 206]. An alternative is to question to governing variables themselves, to subject them to critical scrutiny (e.g. to emphasize open inquiry of the anxiety rather than trying to suppress it). "*Double-loop learning occurs when errors are corrected by changing the governing variables and then the actions*" [2, p. 206]. Argyris and Schön argued that many people espouse double-loop learning, but are unable to produce it, and are unaware of it.

Model I and Model II

Briefly, Model I is composed of four governing variables: (1) achieve the purpose as the actor defines it; (2) win, do not lose; (3) suppress negative feeling; and (4) emphasize rationality [1]. The primary behavioral strategies are to control the relevant environment and tasks unilaterally and to protect one-self and others unilaterally. Thus most used action strategy is unilateral control over others. Characteristics ways of implementing this strategy are making unillustrated attributions and evaluations (e.g. "your work is poor"), advocating courses of action in ways that discourage inquiry (e.g. "surprise me, but don't take risks"), treating one's own views as obviously correct, leaving potentially embarrassing facts unstated [1]. The consequences are likely to be defensiveness, misunderstanding, and self-fulfilling and self-sealing processes [2]. Model I leads to low-learning and double-loop learning does not tend to occur. Argyris and Schön look to move people from a Model I to a Model II that fosters double-loop learning.

The governing variables of Model II include (1) valid information, (2) free and informed choice, and (3) internal commitment: vigilant monitoring of the implementation choice to detect and correct error [2]. The behavioral strategies involve sharing control with those who have competence and who participate in designing or implementing the action [1]. As in Model I, prominent behaviors are advocate, evaluate, and attribute. Unlike Model I behaviors, Model II behaviors stem from action strategies where attributions and evaluations are illustrated with observable data, and the surfacing of conflicting view is encouraged in order to facilitate public testing of them. The consequences include minimally defensive interpersonal and group relationship, high freedom of choice, and high risk taking. Defensive routines are minimized and genuine learning is facilitated [1, 2].

References

[1] C. Argyris, R. Putnam, and D. McLain Smith, "Action Science, Concepts, methods, and skills for research and intervention", San Fransisco: Jossey-Bass, 1985, pp. 4, 80.

[2] C. Argyris, "Double-Loop Learning, Teaching and Research", Learning & Education, Vol. 1 (2), Dec. 2002, pp. 206-219

Figure 2. Theory of Action by Chris Argyris and Donald Schön.

C. Application to Small Projects and to Young Engineers

1) Espoused theories and theories-in-use

Our first observation is that, in the software engineering field - and especially in small software projects – the horizon of standards of processes and practices such as 12207and 15504 standards may constitute the espoused theory, since it is what engineers claim to follow. But what engineers do

(and this action is designed and do not "just happen") may reveal a different theory-in-use. Espoused theory and theory-in-use may be consistent or inconsistent. Young engineers are generally neither aware of her/his theory-in-use nor of aware any inconsistency although experienced engineers may be. Theories-in-use can be made explicit by reflecting on action [1]. We believe that adopting the reflective practitioner methodology may help young engineers in the development of software systems. We fully agree with Hazzan and Tomayko by adopting a reflective practitioner approach to Software Engineering education: "*students' ongoing reflection of the process of developing software systems becomes part of their software development process and consequently, students may improve their understanding of the essence of the methods that guide software creation processes*" [16].

Related work. Hazzan and Tomayko presents in [16] a course intended to develop reflective thinking into the education of software engineers. Lesson 8 is about learning processes in software engineering; they discuss from the reflective practice and its relevance to Software Engineering but theories of action are not evoked.

2) Small projects as a learning organization

Regarding Argyris and Schön theories, our second concern is about the structure of a Very Small Enterprise (VSE, up to 15-25 employees), the different roles of its employees and the VSE behavior as an organizational learning system. Authors both worked as project managers for nearly ten years in a small department of a software services company. The organization was typical of a Very Small Enterprise: one manager acting as a project manager, three other software project managers (typically from 2 to 4), between 8 and 12 software developers. The only difference with a VSE was the existence of a corporate baseline – but far unknown of most department employees. As pointed out in the introduction, the environment of our apprentices either plunged in their virtual practicum at the university or in their work situation during the work placement period, is very similar to a VSE or a small software department. Applied to a small structure acting as a learning organization, a model of its theory-in-use can be built from the elements presented at the beginning of this section.

Our hypothesis is that the whole team acts as a learning organization with a theory-in-use mastered by Model I or II. The *governing variables* are those of Model I or II; project managers (and any employee with useful knowledge and skills) design and implement *action strategies*. Everyone in the VSE experiments *consequences*, intended or unintended, productive or counterproductive. According to this hypothesis, when a VSE uses Model I as a master program of its theory-in-use; double-loop learning is inhibited, error escalates and effectiveness in problem solving and in execution of actions tends to decrease [17]. But, when individuals move toward a Model II program, the whole organization will operate as an Organizational II learning system – a rare phenomenon [17].

In the new work placement system, the problem of the learning organization is pregnant for several reasons. First, the discontinuity of university periods (9 periods of 2 weeks) requires a solid learning organization to avoid waste of time. Secondly, although industry periods are focused on productive activities and under the control of industrial managers, they are associated to learning objectives and academics require a kind of distance control on the constructive activities that happen on these occasions. Moreover, individual or collective industrial practices that apprentices trust as efficient have to be re-used in the practicum because they form an important part of their experience and theory-in-use. Anchoring each student's individual learning path with his/her industrial experience and exploiting these experiences in a learning organization system is a crucial issue.

Previous and related work. In the previous system, we did not pay attention to this challenge. Students were plunged during 7 full-months in the reflective practicum, closely coached during the first 4.5-months iteration, still accompanied in the second iteration where they have to produce autonomously [18]. As they shifted from apprenticeship to production, students naturally reused the reflective practices employed in the first iteration and a kind of learning organization (probably a hybrid of Model-I and II) spontaneously occurred.

Halloran [19] investigates the relationship between a software process assessment and improvement model and organizational learning. The case study observes the introduction of SPICE [20] (ISO/IEC 15504) as a Software Process Assessment and Improvement program in an Australian multimedia company of approx. 70 employees. The paper points out the difference between "engineer's espoused theory" and his/her "theory in use" but it does not develop this matter and rather focuses on the use of organizational learning to promote a proactive approach culturally to continuous improvement and learning procedures. In the case study, Halloran attributed the limited success of the SPA & SPI program to the fact that most of the "learning" associated with this program was single loop learning [1] and recommended to change this model.

3) Process assessment

Regarding the understanding of software processes that students are building, we were faced to a crucial issue. In the previous system (without work placement), students learned software processes by doing during the first iteration and reproduced these processes during the second one. Thus, links were easy to establish and a practical understanding of software processes occurs. Now, first iteration (focused on constructive activities) and second iteration (focused on productive activities) are performed on different projects. The former is an apprenticeship project driven by the university and the latter is an industrial project driven by the companies with whom students are placed. We need an assessment framework common to both projects that allows apprentices to relate and cumulate experiences.

The two authors have defined in [21] that most productive activities have a constructive part that the human subject performs in order to build and develop competencies. We are using assessments fully integrated with the life cycle, constituted by the reader/author feedback cycle, progress meeting and peer reviews. We call it regulation assessments

referring to De Ketele "[…] an open process whose priority function is to improve the working order […] of a part or of the whole system" [22].

Although we think that process assessment as defined in ISO/IEC 15504 [13] or CMMI is out of reach of young engineers and VSEs, we believe that a personal Process Assessment Model and a simplified Process Reference Model are necessary to provide a knowledge basis needed for the practice of software engineering. Furthermore, we think that these models may provide a model of theory-in-use, providing students with a link between apprenticeship and work experiences.

Related work. Von Konsky and Ivins [23] propose an approach for assessing the capability and maturity of undergraduate software engineering projects. The approach is based on an adaptation of the Capability Maturity Model Integration and a hybrid version of the Team Software Process. "*The approach was shown to focus student attention on process improvement and on the attainment of realistic and measurable project management goals*" [23]. Our goal differs because we focus on making explicit theory-in-use.

III. SEAMING THE PROJECT WITH REFLECTIVE PRACTICES

A. Academic assessment

Assessment of student learning has to be based on making clear distinctions among types of learning outcomes. Classical types include individual knowledge, process skills, and products. In our system, individual knowledge is assessed essentially through individual reports and viva voice examination. Products (and indirectly processes involved) are assessed with assessment procedures mimicking industrial usages; these assessments consist of evaluations based on the review of apprenticeship stages and on the qualification of software products. Ideally, apprentices' understanding of processes should take the form of a reflective written essay intended to explain and improve upon their software processes, along with teamwork and communication used to support their performance in performing the processes.

Final reports with in-class presentation may be used for reflective practices but as they are assessed with a mark, reports are biased with the students' assessment strategy. We found more useful to provoke intermediary reports without any assessment. Of course, students will reuse analysis, writings and oral presentation in their final reports and for required self-assessments (it may be a supplementary motivation to perform sound intermediary reports) but we minimize assessment biasing.

During the industrial periods, apprentices have to record events of interest in an individual journal associated with significant artifacts they may have used or produced. Each two months (corresponding to two industry periods of 2 weeks each), the academic period begins with a half-day where each apprentice (12 at all) presents an intermediary report of his/her activities at work. S/he has ten minutes to make an oral presentation in front of other apprentices, followed by 10 minutes of question from coaches and eventually other apprentices. Writing a meeting report, called a sandwich report, is assigned to two students based on individual report provided by each apprentice. In order to prepare these meetings, apprentices generally work from the industrial recording mentioned above.

During academic periods without industrial reporting, apprentices have to perform a first-level process analysis on the state of software processes of their academic project. Each apprentice has to work on two or three processes (13 used at all) and to build an intermediary process element called Step-of-action based on historical Course-of-action Units related to a given process (cf. §II.A and Fig.1). This analysis is intended to produce a re-composition of the global dynamic in terms of smaller units and relations of sequencing and embedding between these units.

B. Regulation assessment

1) Apprentice – coach dialogue

Each scene gives rise to one or several deliverables. When the product is delivered, the coach carefully examines it and writes an assessment card including a general assessment together with all the points to improve or to start over. When apprentices enquire into assessment cards, first reactions are generally defensive. Worried about avoid failure, and embarrassing or threatening feelings, apprentices' action strategy tend to protect themselves and to control work to be redo unilaterally, applying strictly each of the coach remark and perceiving coach's feedback as orders. When the apprentices get used with discomfort and fearful of being vulnerable, and when s/he understands that defensive routines are limiting her/his ability to understand and to develop oneself, it becomes possible to make a step towards Model II. The feedback is given in front of the authors, it allows authors to deepen, to discuss, eventually to contest remarks made by the coaches. Following this briefing, students have to update or start again their product, which will be assessed again.

2) Peer review

Within industry, the objectives of a peer review are to detect and to eliminate, early and efficiently, the defects of products under development. Each undetected defect during a phase will induce its propagation in the later phases and will require ulterior supplementary work.

In our system, we adapted and extended the industrial peer review in order to provide a support for skills transfer between apprentices of the same team. This kind of peer review privileges the collecting of questioning, additional information, improvement proposals rather than the collecting of potential defects even if that implicitly happens. This data collecting encourages exchanges between students, helps to take some improvement proposals into account and aids to correct major defects in the resulting products of an apprenticeship stage. When the underlying governing values of this cooperative work are those of Model II (valid information, free and informed choice, and internal commitment, cf. Fig. 2), apprentices' action strategies combine advocacy and inquiry. Attributions and evaluations are illustrated with relatively directly observable data, and the surfacing of conflicting views is encouraged in order to facilitate public testing of them. Hence, learning is enhanced.

C. Support practices

1) Activity tailoring

Any software development baseline needs to be tailored to each project. This tailoring process defines the activities to be performed and products to be developed and delivered.

In our learning process, we transform this tailoring concept in an "activity tailoring" intended to encourage reflection on action. Tailoring an activity can be assimilated to a preliminary work of thought and suggestion for how to proceed in order to perform the concerned activity. For example, using a new method or tool begins with an exploration in order to tailor its usage to the specificities of the project. Indeed when a production task is hard to understand such as design (that requires a real experience), it may be preferable to think "upstream" and to perform the design of a program, such a technical architecture prototype, subsequently to its realization issued in a previous stage of the learning process.

Each tailoring ends with the writing of an usage guide or implementation guide of the concerned activity. This kind of tailoring activity favors and encourages students' initiatives and creativity on technical, methodological aspects or any activity of the software development process.

When the student's repertoire is empty for a given activity, any new task seems impossible. Activity tailoring provides a bootstrapping of activity. After tailoring, faced with the intended activity, even it is perceived as new and may be threatening, it minimizes defensive reasoning, it provides an higher freedom of choice.

2) Retro-engineering

In the pedagogical field, retro-engineering is an inductive approach. It is the reconstruction from back to front of a process, starting from the result of an activity. "[…] the retro-engineering approach or intuitive engineering […] means analyzing, breaking down an activity step by step in order to ask how it is used in a given situation. This approach is applied each time we need to understand "how it is working" or "how it is made"" [24, citing Pinker].

The idea is to confront students with an existing system whose techniques or associated development environment they do not know. The aim is to acquire skills that will allow the maintaining and the evolving of the system. The approach follows a standard framework: 1) Install, run and study the system; 2) Install and configure the development environment that will be used to modify the system; 3) Perform a prototype satisfying the same technical constraints with the permanent obligation for students to observe themselves at work; 4) Write a document for team mates so that they can understand and continue the prototype.

The retro-engineering of an activity adds iterative and incremental experiences. When this activity is performed in a small group, behavioral strategies should share control with those who have competence.

D. Discussion

This section has presented some points of the immersion system intended to contribute to the software practitioner education. These practices are available for the coach and may be incorporated into or linked with an apprenticeship situation. Students do not perceive these reflective activities as software engineering activities but they do not perceive them as a constraining pedagogical tool whose finality would be to be reflective "at any cost". They take them as comforting steps which allow them to take a proper view and stabilize their knowledge. They rapidly use them because they see their immediate interest in order to deal rationally with the problem they have. They do notice that these activities contribute improving their own apprenticeship process, the production process, also indirectly, and finally the quality of the expected system.

IV. A Journal of the Course of a Project

A. An empirical case study

This case study is based on the activity of a team of 6 young software engineering apprentices with the two authors acting as observers, .This case study observes the whole course of the project. As pointed out by Singer and Vinson [41], apprentices' consent is required and apprentices agreed to participate.

The project is a semantic annotation tool. The main goal of the project is to provide a semantic annotation tool able to annotate (indexing through metadata) Web resources, search (on metadata) in different modes, browse (hierarchically or with facets), manage RDF vocabularies (semantic schemas), and deal with the scope of annotations (public or private). The project uses Jena - http://jena.sourceforge.net/ an open-source Semantic Web programmers' toolkit - as RDF API.

B. Recording the project progress

As the project progress, events of interest are recorded in a journal associated with significant artifacts they may have used or produced. As described in III.B.2, each individual course-of-action is accounted, on a 2-3 days basis, in an instance of the smaller unit, that we called a Performed Activity. Apprentices create a wiki page for each individual activity performed during the stage, fill this page with a short description of activities performed, link this page with related other pages (scene, person, artifact), and upload artifacts in the wiki. At the end of each stage (two weeks), apprentices account individual and collective work in the finest grain of collective course-of-action, called a Course-of-action Unit, which organizes several individual Performed Activities. This accounting provide a first-level of reflection-in-action.

Each two months, apprentices have to perform a first-level course-of-action analysis on the state of software processes of their academic project. Each apprentice works on two or three processes and builds an intermediary process element called Step-of-action based on historical Course-of-action Units related to this process. This analysis is intented to develop reflection-on-action.

All information is recorded in two semantic wikis:

- http://oysterz.univ-brest.fr/12207, the 12207 wiki is a hypertext reference of the ISO/IEC 12207:2008.
- http://oysterz.univ-brest.fr/company, the company wiki contains Processes group / Processes / Exemplar Activities

and Stages / Scenes decompositions but its most important part is the journal recording the project progress. Its structure is based on the model given in the lower-half of Fig. 1.

C. Accounting

This project has completed its 9 stages and quantitative facts are given in table II – that is the number of instances (wiki pages) in each category. For each process, we have the quantity of apprenticeship Scene (SCE), the quantity of Performed Activity (PAY: individual), the quantity of Course-of-action Unit (CAU: collective), the quantity of Step-of-action (STE: higher-level construct). The 6th column gives an indication of the quantity of Software Engineering Activities that may be envisaged in the process. 7th (Act) and 8th (Tsk) report the 12207 decomposition of related processes: number of activities (Act) in (the) process(es) and number of corresponding tasks (Tsk). The 9th and last column gives the number of Base Practice (BP) in the corresponding process of the 15504 standard.

TABLE I. QUANTITATIVE FACTS

Process	***SCE***	***PAY***	***CAU***	***STE***	***SE Act.***	***Act***	***Tsk***	***BP***
Project management	13	22	13	3	5	7	14	15
Quality insurance	2	2	1	1	2	4	16	8
Configuration management	2	2	2	2	3	6	6	10
Requirements capture	10	18	10	0	5	5	12	6
Software analysis	2	2	2	2	2	1	3	6
Technical architecture	7	10	5	3	4	2	2	-
Software design	7	9	5	4	4	2	15	12
Software construction	8	16	4	3	5	1	5	4
Integration – validation	8	12	5	5	5	6	20	20
Technical support	8	16	2	2	2	3	4	6
Methods and tools support	3	3	3	2	2	-	-	6
Documen - tation	2	4	2	1	2	4	7	8
Installation - deployment	1	N/A	N/A	N/A	2	2	5	6

As noted in §III.B.1, students reports their activity at the end of each stages (2 weeks). When an activity temporally crosses over stages (e.g. technical support or coding), students adopted a simple strategy: they created one single mid-level structure (Course-of-action Unit) and linked to individual low-level units (Performed Activity) belonging to different stages.

45 Steps-of-action are envisaged. Simple processes as Design are correctly reconstructed. There are complex processes that may be oversimplified in the practicum (e.g. Configuration Management or V&V) and the reconstruction is correct regarding the simplified process but it is partly inaccurate. For some complex processes involving many scenes (e.g. Requirements), the reconstruction failed, probably because apprentices are unable to perceive an abstract view of the process.

D. Discussion

Table I help to figure a quantitative view of the journal writing activity. The amount of 12207 tasks (or 15504 Base Practices as well) give an indication over the density of the process. The higher are these amounts, the higher is the complexity; this it should lead to a process reconstruction involving a higher amount of Step-of-action related to a roughly amount of Software Engineering Activities. A major difference between col. 5th (STE) and col. 6th (SE Act.) may indicate that the reconstruction failed.

On the other hand, amounts of Performed Activity (PAY) and Course-of-action Unit (CAU) are closely depending on the pedagogical objectives and on the organization that the coach put on the process. A high amount of Scene / Performed Activity / Course-of-action Unit may represent the complexity of the process but may also indicate that the process is a supporting process whose activities are performed all the life cycle along.

V. CONCLUSION AND PERSPECTIVES

Argyris and Schön developed a model of the processes involved in theories-in-use based on three elements: governing variables, action strategies, and consequences [2]. Then, in [26] they explored the nature of organizational learning and defined two kind of learning: simple-loop learning and double-loop learning. Then they set up two models (Model I and Model II) that describe features of theories-in-use that either inhibit or enhance double-loop learning.

Model I theory-in-use requires defensive reasoning. Unlike the defensive reasoning, the logic used in Model II is not self-referential. As a consequence, defensive routines that are counterproductive to learning are minimized, and genuine learning is facilitated. Thus, an ultimate goal of our education system is to promote Model II as a theory-in-use.

Two hypotheses are discussed (1) that theories-in-use can be made explicit by reflecting on action; (2) that the whole team acts as a learning organization with a theory-in-use mastered by Model I or II. As a case study, the activity of a team of 6 young software engineers accompanied with two participants-to-observe is shown.

The current state of this work let suggest that (1) a personal Process Assessment Model and a simplified Process Reference Model may form an initial structure of the theory-in-use and support reflective thought (2) when each team member is going from a “pre-reflective consciousness” towards a reflective attitude, the whole team may act as a learning organization.

ACKNOWLEDGMENT

The authors wish to thanks François-Xavier Bru, Gaëlle Frappin, Ludovic Legrand, Estéban Merrer, Sylvain Piteau, and Guillaume Salou for their participation to this work.

REFERENCES

[1] C. Argyris, and D. Schön, “Theory in practice: Increasing professional effectiveness”, San Fransisco: Jossey-Bass, 1974.

[2] M. Finger, and S.B. Brand, “The concept of the “learning organization applied to the transformation of the public sector” in: M. Esaterby-Smith, L. Araujo and J. Burgoyne (eds.) Organizational Learning and the Learning Organization, London: Sage, 1999.

[3] J. E. Tomayko, “Carnegie Mellon's software development studio: a five year retrospective” in Proceedings of the 9th Conference on Software Engineering Education, New York: IEEE Computer Society Press, pp. 119-129, 1996.

[4] ISO/IEC 12207:2008, “Information technology -- Software life cycle processes”. Geneva: International Organization for Standardization (ISO), 2008.

[5] D. Schön, “The Reflective Practitioner”, New York: Basic Books, 1983.

[6] D. Schön, D., “Educating the Reflective Practioner: Toward a New Design for Teaching and Learning In the Professions”, San Fransisco: Jossey-Bass, 1987.

[7] S. Kuhn, “The software design studio: an exploration”, IEEE Software, Volume 15 (2), March-April 1998, pp. 65-71.

[8] P. A. Laplante, “An Agile, Graduate, Software Studio Course”, IEEE Transactions on Education, Vol. 49 (4), Nov. 2006, pp. 417-419.

[9] ISO/IEC 12207:1995, AMD 1:2002, AMD 2:2004, “Information technology -- Software life cycle processes”, Geneva: International Organization for Standardization (ISO), 1995, 2002, 2004.

[10] R. Samurçay, and P. Rabardel, “Work competencies: some reflections for a constructivist theoretical framework” in Proceedings 2nd Work Process Knowledge Meeting: Theoretical approaches of competences at work, Courcelle sur Yvette, 1995.

[11] ISO/IEC FCD 24765, “Systems and software engineering – Vocabulary”. Geneva: International Organization for Standardization (ISO), 2009.

[12] P. Roques, and F. Vallée, “UML en action”, Paris: Eyrolles, 2002.

[13] ISO/IEC 15504:2004, “Information technology -- Process assessment”. Geneva: International Organization for Standardization (ISO), 2004.

[14] D. Boud, “Using Journal Writing to Enhance Reflective Practice”, New Directions for Adult and Continuing Education, Vol. 90, Summer 2001, pp. 9-17.

[15] C. Argyris, “Organizational learning and management information systems”, ACM SIGMIS Database, Vol. 13 (2-3), Winter-Spring 1982, pp. 3-11, ISSN:0095-0033

[16] O. Hazzan, and J.E. Tomayko, “Reflection processes in the teaching and learning of human aspects of software engineering”, in Proceedings of 17th Conference on Software Engineering Education and Training, New York: IEEE Press, pp. 32- 38, 2004, doi:10.1109/CSEE.2004.1276507

[17] C. Argyris, and D. Schön, “Organizational learning : A theory of action perspective”, Reading: Addison Wesley, 1978

[18] V. Ribaud, and P. Saliou, “Software Engineering Apprenticeship by Immersion”, in Proceedings of International Workshop on Patterns in Teaching Software Development, ECOOP’03, Darmstadt, 2003.

[19] P. Halloran, “Organisational Learning from the Perspective of a Software Process Assessment & Improvement Program” in: 32nd Hawaii International Conference on System Sciences. New York: IEEE Press, 1999.

[20] Software Process Improvement and Capability dEtermination (SPICE), Software Process Assessment - Version 1.00, http://www.sqi.gu.edu.au/spice/docs/baseline, 1995

[21] P. Saliou, and V. Ribaud, “Former un praticien réflexif de l’ingénierie du logiciel (The training of a reflective practitioner in software engineering)” in Colloque Les pédagogies actives : enjeux et conditions, pp. 535-545, Louvain: University of Louvain , 2007.

[22] J.M. De Ketele, and X. Roegiers, “ Méthodologie du recueil d’informations”, Bruxelles: De Boeck, 1993.

[23] B. R. von Konsky, and J. Ivins, “Assessing the Capability and Maturity of Capstone Software Engineering Projects”, Research and Practice in Information Technology, Vol. 78. S. and M. Hamilton (eds.), 2008.

[24] P. Dessus, “La planification des séquences d’apprentissage, objet de description ou prescription ? ”, Revue Française de Pédagogie, Vol. 133, pp. 101-116, 2000.

[25] D. Schön, “Educating the Reflective Practitioner” in Meeting of the American Educational Research Association, 1987.

[26] O. Hazzan, “The reflective practitioner perspective in software engineering education”, Journal of Systems and Software, Vol. 63 (3), September 2002, pp. 161 – 171, ISSN:0164-1212

[27] J. Leplat, “Regards sur l'activité en situation de travail - Contribution à la psychologie ergonomique”, Paris : Presses Universitaires de France, 1997.

[28] F. Morandi, “Pratiques et logiques en pédagogie”, Paris: Nathan, 2002.

[29] M. Huo ,H. Zhang, and R. Jeffery, “An exploratory study of process enactment as input to software process improvement” in Proceedings of the 2006 international workshop on Software quality table of contents, pp. 39 - 44, New York: ACM, 2006, ISBN:1-59593-399-9

[30] J. Singer, and N. G. Vinson, “Ethical issues in empirical studies of software engineering”, IEEE Transactions on Software Engineering, Vol. 28 (12), Dec 2002, pp. 1171- 1180, doi:10.1109/TSE.2002.1158289

[31] J. Topping, Sandwich courses, Phys. Educ. Vol. 141 (10), 1975, pp. 141-143, doi:http://iopscience.iop.org/0031-9120/10/3/003