Equipping Software Engineering Apprentices with a Repertoire of Practices

Vincent Ribaud, Philippe Saliou

Equipping Software Engineering Apprentices with a Repertoire of Practices

Vincent Ribaud, Philippe Saliou
Université de Bretagne Occidentale, LISyC - EA 3883
Université Européenne de Bretagne
Brest, France
{Vincent.Ribaud, Philippe.Saliou}@univ-brest.fr

***Abstract*—Argyris and Schön distinguish espoused theories - those which people speak about – from theory-in-use - those which can be inferred from action. In small software teams, developing reflective thinking about action is a vital necessity in coping with change. We address these issues in a Masters of Software Engineering, performed with an alternation between university and industry. University periods are dedicated to a long-term project performed in a reflective practicum. It aims to develop a repertoire of practices which helps young engineers deal with the 'messiness' of situations. Such a practicum provides students, working in groups, with the possibility of reflecting on action. We propose using the Course-of-Action framework to record observable aspects of the actor's activity into semantic wikis. Two hypotheses are discussed (1) self-analysis and self-assessment help to reveal theories-in-use; (2) the Course-of-Action observatory helps maintain awareness of the repertoire. A case study of a 6-apprentice team illustrates the observatory use and the reconstruction of apprentices' activity. Primary conclusions are that self-observation and self-analysis of a software engineer's activity help raise awareness of the initial structure of the repertoire. We are however unable to conclude that it helps reveal their theory-in-use (what governs an engineer's behaviour) - usually tacit structures.**

Keywords-component; reflective practitioner, software engineering processes, Course-of-Action, semantic wiki.

I. INTRODUCTION

This paper is an extended and enhanced version of a paper presented at the ICCGI 2009 conference [1].

Small organizations – and small software teams especially – need to constantly adapt their task force to the products or services to be delivered. The software process community shares a tacit axiom that improving software processes automatically improves software products and contributes to the project success. Many efforts have been made to extensively define a set of processes and build assessment methods intended to verify to what extent defined processes are performed.

Yet D. Schön [2] argued that experienced professionals deal with the 'messiness' of practice not by consulting the research knowledge base, but by engaging in 'reflection-in-action': experiencing surprise in a new situation and responding to surprise through a kind of improvisation. To educate the reflective practitioner, Schön recommended looking at traditions of education for artistry – in art studios, or in conservatories of music and dance. Schön qualified these students as learning by doing in a reflective practicum. This analogy was used to provide a suitable educational environment for software design at CMU [3] or at MIT [4].

The notion of repertoire is very important in Schön's approach. Practitioners build up a collection of ideas, examples, situations and actions. "*A practitioner's repertoire includes the whole of his experience insofar as it is accessible to him for understanding and action*" [5, p.138].

This hypothesis – coupled with the observation that students (and young engineers) are experiential, tending toward learning by doing rather than listening – led us to focus on the goal of providing software engineering graduates with a non-empty repertoire of practices, together with an operational knowledge of software processes, activities and tasks. In 2002, we built an education system called 'Software Engineering by Immersion' entirely based on performing complete development cycles of a software project, and accomplished in three iterations. This 3-iterations system can be summed up with the sentence: 'A first turn to learn by doing, a second turn to do autonomously what has been learned, and a last turn to work effectively in a business. A Process Reference Model - greatly simplified from the ISO/IEC 12207:1995 standard and its amendments [6] - was used as the initial structure of the repertoire. As realistic working situations were experienced, students were provided with progressive filling of their repertoires.

In 2007, local employers in Brest requested employees in 'sandwich' (or work placement) conditions, and we adapted the 'Software Engineering by Immersion' programme to run as a work placement course. In such a programme, some of the educational objectives and related assessments are devoted to periods in industry. The second and third iterations were good candidates to assign to industrial periods, and first iteration (at the university) and second iteration (in the industry) were organised into alternating 2-week periods. We do however face the problems of relating the university-based and industry-based elements of the student's experience and avoiding a situation in which learners are required to climb two ladders simultaneously.

We decided to redesign the repertoire (including its construction and 'filling') in order to - as far as possible – meet the twin challenges of learning and producing within a small software project. Our current proposition is to use two theories of action, the former from Argyris and Schön [7] about theory-in-use and espoused theory and the latter - the Course-of-Action framework pioneered by Theureau and Pinsky [8]. The main idea is to provide young engineers and

small projects with an observatory of their individual and collective activity (by observable, we mean what is presentable, accountable and commentable) and to instigate the reconstruction of a high-level view of the global Course-of-Action from small and individual units of action.

The nature of collected data is very different and subject to change, as is the semantic of the relationships between data. This addresses technical challenges related to content management. We choose to use semantic wikis as a lightweight authoring platform. As Maxwell [9, p.199] outlines "*Our experience with and reflection on using wiki as a platform suggests that there is much to be gained from an approach which builds up from simple foundations rather than attempting to customize already-complex architecture*".

Section 2 overviews theories of this introduction, their application to software engineering and some related work. In Section 3, we present courses-of-action for software apprentices. Observing the course of apprentices' projects is discussed in Section 4. In Section 5, we present some data excerpts of a case study. We conclude the paper with a discussion and perspectives.

II. RESEARCH ISSUES AND RELATED WORK

We will present Argyris and Schön's theories of action as well as certain elements of the Course-of-Action framework. We will present two research hypotheses, and related work.

A. Espoused theories and theories-in-use

A starting point of Argyris and Schön's [7] theory (see Figure 1) is that people design action to achieve intended consequences, monitoring themselves in order to learn whether their actions are effective. They made a distinction between two contrasting theories of action: *theories-in-use* and *espoused theories*. "*When someone is asked how he would behave under certain circumstances, the answer he usually gives is his espoused theory of action for that situation. This is the theory of action to which he gives allegiance, and which, upon request, he communicates to others. However, the theory that actually governs his actions is his theory-in-use*" [7, pp.6-7].

Our first observation is that, in the software engineering field, lifecycle processes standards such as the 12207 [10] and process assessment standards such as 15504 [11] or CMMI may constitute the espoused theory, since it is what engineers claim to follow. But what engineers do (and this action is designed - it does not 'just happen') may reveal a different theory-in-use. A young engineer is rarely aware of either their theory-in-use or of any inconsistency - although an experienced engineer may be.

Theories-in-use can be made explicit by reflecting on action [7]. According to Schön, reflective thought takes place in a reflective practicum. Schön advocated traditions of education for artistry as exemplar through their reflective practicum. "*[…] its main features are these. It's a situation in which people learn by doing, […] where they learn by doing in a practicum which is really a virtual world. A virtual world in the sense that it represents the world of practice, but is not the world of practice […] in that world, students can run experiments cheaply and without great danger […] in interaction with someone who is in the role of coach*" [12].

A reflective practicum is intended to run experiments and develop reflection-on-action. In our practicum, we use organized processes to drive project and competencies building. Parallel to the engineering activities required by the project, apprentices are regularly required to self-analyze and self-assess their engineering practices. Our first hypothesis (H1) is that self-analysis and self-assessment helps an apprentice to reveal their theory-in-use.

Previous and related work. The studio is the central training method in architecture schools and this analogy was used to provide a suitable educational environment for software [3] [4]. Our system is very close to Tomayko's work, and most of his observations apply to our system: "*The use of a well-established development process, a matrix organization, and one-to-one mentoring give the highest return on investment*" [3, p.119].

Hazzan and Tomayko present in [13] a course intended to develop reflective thinking about the education of software engineers - but theories of action are not evoked.

Halloran [14] investigates the relationship between a software process assessment and improvement model and organizational learning. The paper points out the difference between 'engineer's espoused theory' and their 'theory in use' but does not develop this idea, focusing instead on the use of organizational learning to promote a proactive approach to continuous improvement and learning procedures.

Models of theory-in-use

Argyris and Schön argued that, even though espoused theories vary widely, theories-in-use do not. They labelled the most prevalent theory-in-use *Model I* and argued that this model inhibits learning. *Model II* favours it. This model looks to three elements. *Governing variables* are values that actors seek to satisfy [1]. Each governing variable can be thought of as a continuum with a preferred range (e.g. not too anxious, yet 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 use 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. feeling better after sporting 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 (Model I). "*Single-loop learning occurs when errors are corrected without altering the underlying governing variables*" [2, p. 206]. An alternative is to question governing variables themselves (Model II), to subject them to critical scrutiny (e.g. to emphasize open inquiry into 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.

References

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

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

Figure 1. Theory of Action by Chris Argyris and Donald Schön.

B. Building their own repertoire

The Course-of-Action theory, pioneered by Theureau and Pinsky [8], provides a framework for analysis of the collective organization of the multiple courses of action in a complex, autonomous and open system. A 'Course of Action' is: "*what, in the observable activity of an agent in a defined state, actively engaged in a physically and socially defined environment and belonging to a defined culture, is pre-reflexive or again significant to this agent, i.e. presentable, accountable and commentable by them at any time during its happening to an observer-interlocutor in favourable conditions*" [15, p.7]. The course of action can be described from two complementary perspectives: from the point of view of its global dynamics - characterizing the units of the course of action and the relations of sequencing and embedding between these units, or from the point of view of its local dynamics, characterizing the underlying structure of the elementary units [15]. Given that we seek to establish a fairly high-level model of actions, we focus on the global point of view because it emphasizes the articulation of work situations and their co-ordination, and is better suited to process-level analysis.

Argyris and Schön suggested that each member of an organization constructs his or her own representation or image of the theory-in-use of the whole [16, p.16]. What is intended is to connect the individual world of the practitioner up with the collective world of an organization. But, prior to this discussion, we need to understand how we perceive our internal structure. The notion of repertoire is a key aspect of Schön's reflection in and on action. Practitioners build up a collection of ideas, examples, situations and actions. "*When a practitioner makes sense of a situation he perceives to be unique, he [she] sees it as something already present in his [her] repertoire. […] It is, rather, to see the unfamiliar, unique situation as both similar to and different from the familiar one, without at first being able to say similar or different with respect to what. The familiar situation functions as a precedent, or a metaphor, or an exemplar for the unfamiliar one*" [5, p.138].

A coach may help both discover the existence of this repertoire, and fill it, with the assistance of reflective thought. Coaches often answer questions with questions, in most cases, simply rephrasing the question. Our proposal is that small projects should be provided with a device which will act as a mirror for their observable activity, and that privileged moments of self-observation in front of the mirror (without adding too much extra work) should be seamlessly integrated in the course of the project. Our second hypothesis (H2) is that the Course-of-Action observatory helps maintain awareness of the repertoire, facilitating self-assessment and self-analysis.

Previous and related work. Hazzan debates the reflective practitioner perspective in software engineering education and the studio as a teaching method [17], but does not address the subject of the practitioner's repertoire.

The 'Course-of-Action' research framework consists of several empirical and technological research programs [15] in various domains such as work analysis [18] or traffic control [19]. We are not aware of any uses of the Course-of-Action framework in the software field.

III. SOFTWARE APPRENTICES' COURSES-OF-ACTION

We will be monitoring the 'Software Engineering by Immersion' Masters programme, and we will present the Course-of-Action observatory and its application to a software project.

A. The 'Software Engineering by Immersion' Masters Programme

1) Structural aspects of our programme

Our Masters Programme in Information Technology and Software Engineering is a 2-year programme, accessible to Bachelor graduates in Computing or 'back to school' software practitioners. For students enrolled in the Software Engineering by Immersion specialization, securing a 'professionalization contract' is a compulsory requirement. During this 12-month contract, the work placement student is a full-time employee, although also attending university for certain periods. Strictly-speaking in France, 'apprenticeship learning' and 'apprentice' are terms reserved for a longer work placement system, but the sake of clarity, we use the term 'apprentice' in this paper.

Competition for this type of contract is performed during the first 7-month intensive courses. The following 4-months are dedicated to an internship period. For the last year, periods at university have to fit into alternating 2-week periods. The year is divided into two periods, the former (from September to mid-May) with movement between university and company, and the latter (from mid-May to August) with a full-time period at the company.

2) Pedagogical objectives and organization

Of the 43 processes of ISO/IEC 12207:2008 [10], we concentrate on the 19 that are related to the software development cycle, which we have reorganized into 3 groups:

- in the *Software Project Management* Process Group: 6.3.1 Project Planning - 6.3.2 Project Assessment and Control, 7.2.2 Software Configuration Management, 7.2.3 Software Quality Assurance;
- in the *Software Development Engineering* Process Group: 6.4.1 Stakeholder Requirements Definition, 6.4.3 System Architectural Design, 6.4.4 Implementation Process replaced by 7.1.1 Software Implementation Process (and its 6 sub-processes: Requirement Analysis, Architectural Design, Detailed Design, Construction, Integration, Qualification), 7.2.4 and 7.2.5 Software Verification & Validation;
- in the *Software Development Support* Process Group: 6.2.1 Life Cycle Model Management, 6.2.2 Infrastructure Management, 6.4.7 Software Installation – 6.4.8 Software Acceptance Support, 7.2.1 Software Documentation Management.

These 19 processes are renamed (and some are also merged) to give a breakdown of apprenticeships into 3 software engineering process groups subdivided into 13 software engineering processes, together with a set of apprenticeship scenes (roughly associated with software

engineering activities) which provide the learning environment and define tasks. This hierarchical group process/process/scenes model, adapted from the ISO/IEC 12207, is given in Tables I and III and is used as a reference framework for the learning objectives.

From the university point of view, this division is the reference framework, in a diploma-awarding perspective. Group processes are course categories, processes are courses and scenes are sessions.

TABLE I. PROCESS BREAKDOWN

Process Group	Process	12207:2008 Related Processes
Software Project Management	Project management	6.3.1, 6.3.2
	Quality insurance	7.2.3
	Software configuration management	7.2.2
Software Development Engineering	Requirements capture	6.4.1
	Software analysis	7.1.2
	Technical architecture	6.4.3
	Software design	7.1.3, 7.1.4
	Software construction	7.1.5, 7.1.6
	Integration and validation	7.1.7, 7.2.4, 7.2.5
Software Development Support	Technical support	6.2.2
	Methods and tools support	6.2.1
	Documentation	7.2.1
	Installation and deployment	6.4.7, 6.4.8

The main feature of the university periods is to learn software engineering by doing, without any computing course but with a long-term project as the foundation of all apprenticeships. Alternating employees are attending university over 9 periods of 2 consecutive weeks, and work in teams of 6 apprentices to build a complete information system.

The rhythm is based on the lifecycle of a project, organized into stages. Each stage was arbitrarily sized to 2 weeks, due to the constraints of alternation. The cycle is: Stage 0: Warm-up; Stage 1: Project set-up; Stage 2: Requirement capture; Stage 3: Requirement analysis; Stage 4: Design; Stage 5: Software construction; Stage 6: Software construction; Stage 7: Integration and Verification; Stage 8: Qualification and Deployment.

3) Competency Reference Model

While apprentices are currently learning by doing software processes, process assessment will not measure a capability level but (in the best case scenario) a learning capability level. Because apprentices are building competencies, and because some reflective learning is required, we choose to promote self-assessment of personal abilities. In 2006, we set down the abilities (or competencies: "*the ability of a person to act in a pertinent way in a given situation in order to achieve specific purposes*" [20]) that scenes are intended to develop. We tried to answer the questions 'What is the student able to do, once the scene is performed? What are the related knowledge topics?' This analysis gave us a set of abilities for each process (see examples in Table II).

So we kept the 2-level breakdown of our reference framework - the first level being called competency areas (corresponding to process groups) and the second level competency families (corresponding to processes), and we positioned abilities and transversal competencies within these areas. Table II shows abilities and knowledge related to certain representative processes for each process group.

TABLE II. EXAMPLES OF COMPETENCY FAMILIES

Abilities and Skills	Knowledge Topics
Project Management	
• To use an ISO 9001 development baseline • To apply a Project Plan, updating it if necessary • Planning and project progress	* Software life-cycle model * Estimation and follow-up of development of components * Traceability and conformity * Project Plan
Software Requirements Capture	
To mobilize specification methods and tools in a real project: • within an ISO 9001-style baseline • in relation to requirements traceability • to produce a Software Requirement Specification	* Software Requirements Fundamentals: definition, functional and non-functional, quantification. * Requirements capture techniques: interviews, client meeting, statement of work, response to solicitation * Procedures, methods and tools for requirements specification. * Use cases.
Software Design	
• To use design methods and tools (in relation with requirements) to produce design documents: system and software architecture and detailed design • To implement methods and modelling tools of various aspects of a system (architecture and decomposition software, data structure) • To implement J2EE development and technology of associated framework • To implement DBMS concepts, techniques and tools	* Software Design Fundamentals: concepts and principles, design role in a development cycle, top-level and detailed design * Software decomposition configuration item, software component, software unit * Software architecture through different views: conceptual, dynamic, physical, data. * UML diagrams to describe static and dynamic views * Object-oriented design
Methods and tools support	
• To know Software Engineering methods and techniques for the software life cycle • To install, adapt, integrate and maintain software tools • To assist engineers in software deployment • To perform a consulting mission, alone or in a group	* Use case models and formats. * Analysis: patterns and model transformation *Design:architectural prototype, generic design *Configuration management: tools and guides * CASE tools

The complete breakdown (3 areas, 13 families, 48 abilities and 11 transversal competencies) is called the competency reference model for our immersion system [21].

B. The Observatory of Apprentices' Courses-of-action

1) The Course-of-Action observatory

A short definition of a Course-of-Action is "*the activity of one (or several) specific actor(s), engaged in a specific situation, belonging to a specific culture, which is significant for the latter, in other words, that can be related or commented by (or them) at any moment*" [18, p. 2].

The Course-of-Action analysis is based on an observatory which includes continuous observations of the behaviour of action and communication in a work situation as well as different traces of other elements such as interpretations, feelings, and judgments [18]. Although the data produced by these observations only gives access to the surface of interactions, it suffices for understanding the structural coupling between an agent and his or her situation.

The Course-of-Action framework proposes data collection methods which include: observing and recording actors' behaviour methods; methods to keep track of behavioural patterns; and methods of provoked and situated verbalization from actors.

Self-confrontation is a prominent activity in terms of documenting the Course-of-Action. This takes the form of collecting verbal data whilst the activity is actually being carried out and/or in a self-confrontation situation (e.g. in the case of driving, the driver watches a film of their journey (which is systematically recorded) and comments on it to clarify their own actions and events [15]).

Other kinds of verbalization, made by agents during activity analysis (called second degree self-confrontation verbalizations - to emphasize the fact that they are situated in the continuity of self-confrontation itself) are also implemented. Here the agents are placed in the position of observers and analysts, and their verbalizations, whilst not data, do nonetheless constitute their contributions to the analysis of their activity [15].

2) Link with field studies

From the 'data collection techniques' point of view, Lethbridge, Sim, and Singer [22] provide a useful taxonomy. They classify techniques according to the degree of human contact required. We use several 'observational first degree techniques' [22] (with direct access to young engineers), including diaries, think-aloud protocols, observation, and participant observation. Because we focus on self-interpretation by the actors, most recording is done manually by the actors themselves. From the taxonomy point of view, it may appear as a 'second degree technique' [22] (with indirect access to young engineers) because it does not require direct contact between participant and researcher.

Representative artefacts of the job are the outputs of software activities and tasks. The analysis of these artefacts falls in a 'third degree technique' [22] (without access to young engineers).

3) What can be collected in the course of projects ?

Recall the definition of the Course-of-Action in §III.B.1: what, in the observable activity of an agent […] is pre-reflexive or significant to this agent, i.e. (i) presentable, (ii) accountable and (iii) commentable by them at any time whilst it is happening […].

We have three types of observation: (i) presentation, (ii) accounting, (iii) comment. Software workers do not achieve complex technical gestures, or do not have to progress through a detailed procedure. So (i) presentations to an observer are quite difficult to reproduce, and the presentable artefacts that are most notable and representative of the job are the outputs of software activities and tasks.

Verbalization is widely used by the coach within the learning process to scaffold the apprentice's activity: when students ask or when tutors consider it to be necessary, a dialogue between apprentice and tutor about what, why and how the apprentice is doing helps them to carry out the activity. Recording this dialogue would be too complicated; furthermore, it would probably compromise - and possibly even destroy - this learning process.

We therefore focus on accounting and comments. Accounting will replace recordings of engineer behaviour. Products and documentary resources are the main objects of presentation, since they describe the activity's inputs and outputs. The 'historical' context of use of (i) resources and product production must also be recorded. This can be described in terms of events and processes, involving occurrences of agents (people) and artefacts (products and resources) meeting in space (in case of distributed collaboration) and time. In the first instance, we consider the individual courses of action of the various participants. At the next stage, we look at collective action involving parts of several individual courses of action taking place synchronically or sequentially. We need to divide individual Course-of-Actions into smaller units, which we call Performed Activity. Each event of interest must be (ii) individually accounted for in an instance of Performed Activity in relation with the apprentice and artefacts involved. This provides a kind of project diary or journal, and is performed in a wiki by each apprentice as the project goes along.

Provoked verbalizations are replaced with self-confrontation interviews as a way of documenting the constraints and effects of the segment of the actor's activity that is personally experienced. Even the smallest unit of collective Course-of-Action is called a Course-of-Action Unit, which organizes several individual Performed Activities. At the end of each 2-week university period, apprentices have to write a short report (individually, but also collectively if they worked on a group task) about what happened during the period (this is called a 'work diary' in the taxonomy of [22]). Apprentices may complete Performed Activity instances previously created, and must create Course-of-Action Unit instances for activities involving several individual Performed Activities.

For the industrial periods, accounting is performed in a different way. As detailed in Section IV.B.5, young engineers must perform a complete self-assessment, 4 times per year, regarding the competency reference model described in §II.A.3 - which is acting as an ability model for their job as an engineer. In support of these periodic assessments, they have to record events of interest in a portfolio, associating events with significant artefacts they may have used or produced.

With very few exceptions, we observed that information about academic and industrial periods are written in a descriptive style (what they do with linked artefacts, when and where) but gave little or no indication why and for what reason they did it. So, we can conclude that these reports are (ii) accounts and not (iii) comments.

4) What can be commented ?

We need a second-level of reporting intended to encourage comments and reflection-on-action. Final reports with in-class presentation could have been used, but we chose not do this in order to avoid introducing bias, since they are assessed with a mark. We found it more useful to trigger intermediary reports without any assessment. Of course, students will re-use analysis, writings and oral presentation in their final reports, as well as for required self-assessments (and this may provide extra motivation to perform sound intermediary reports) but we minimize assessment bias.

Among the self-confrontation methods used in the Course-of-Action framework, one is the so-called second level self-confrontation interview. It is performed after the self-confrontation interview proper. Its procedure is radically different, because its aim is not to collect empirical data about the actor's experience at instant t, but to develop co-operation in the analysis of the activity between the researcher and the actor [19]. We borrow this practice for reporting on industrial and academic periods.

Every two months (corresponding to two industry periods of 2 weeks each), the academic period begins with a half-day during which each apprentice (12 in all) presents an intermediary report of their activities at work. Writing up a meeting report is assigned to two students, based on the individual reports provided by each apprentice.

During academic periods where there is no industrial reporting, apprentices must perform a first-level analysis on the state of the software processes of their academic project. Each apprentice has to work on two or three processes (13 are used in all), building an intermediary process element called Step-of-Action based on historical Course-of-Action Units related to a given process. This analysis is intended to produce a reconstruction of the global dynamic in terms of smaller units and the sequencing and embedding relationships between these units.

IV. OBSERVING THE COURSE OF APPRENTICES' PROJECTS

We will survey the models used in the 'Software Engineering by Immersion' programme. We will present an enacted project that will be used as a case study.

A. Process models

1) Prescribed work

Leplat [23] has identified a difference between prescribed task and effective task. The prescribed task is a task that a designer or an organizer wishes an operator to perform. What he/she really achieves is the effective task.

It is a hard task to define things to do in a software projects, hence to provide a structured description of the prescribed tasks. Several standards were written with this goal. In our opinion, the process dimension of the ISO/IEC standard 15504:2004 [11] provides a complete view of the prescribed work to be done in a software project.

ISO 15504 separates process and capability levels in two dimensions. In the process dimension, individual processes are described in terms of Process Title, Process Purpose, and Process Outcomes as defined in ISO/IEC 12207 (where each life cycle process is also divided into a set of activities; each activity is further divided into a set of tasks [10]). In addition, the process dimension provides: a) a set of base practices for the process, providing a definition of the tasks and activities needed to accomplish the process purpose and fulfil the process outcomes; b) a number of input and output work products related to one or more of its outcomes; and c) characteristics associated with each work product [11]. The capability dimension consists of six capability levels (Level 0 reflects an incomplete process) and the process capability indicators for nine process attributes for levels 1 to 5. A process attribute is "*a measurable characteristic of process capability applicable to any process*" [11, p. 4]. Figure 2 represents the two dimensions and a performance of process assessment.

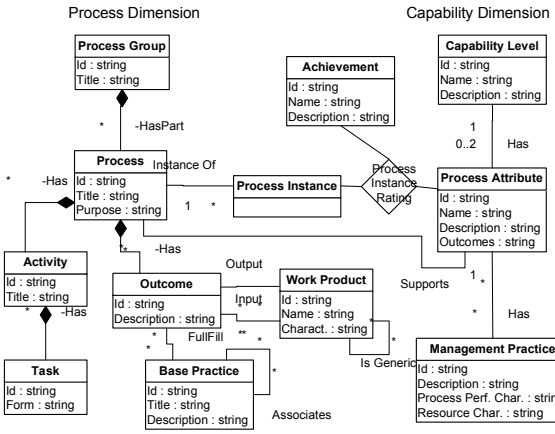

Figure 2. 12207 and 15504 Reference Models. Performing a process assessment yields a rating for each process attribute. A rating is a judgement of the degree of achievment (None, Partially, Largely, Fully) of the process attribute for the instance of the assessed process.

2) Work scenes

The 15504-5 standard provides software engineers with an exemplar model of a software project. Unfortunately, such an exemplar model is necessary but not sufficient for learning purposes. In our system, we have to organize the apprentices' activity into small units of work called an apprenticeship/production scene.

The apprenticeship/production scene is the reference context in which a part of the play happens: the scene aims for a unity of place, time and action; the scene is at once a situation in which people learn and do; a scenario of actions; a role distribution, and an area mobilizing resources and means. The different components of a scene, along with their articulation are depicted on a card. The card structure is standardized (see an example in Figure 3).

The main elements of a card are the process group / process (here development / design) tied up with the work; the role to play (here, designer or architect) with team-mates' assignment; the work description (here, the detailed design); the products (deliverables) to deliver (here, a Software Design Document, SDD); the supplied pedagogical resources (here, a writing guide, real SDD samples and an analysis and design course); workload and lead-time information.

No. 24	**Date**:	**Origin**:	**Roles assignment**	
Project:	**SCENE CARD**		Designer Architect	Solenn Arnaldi Aude Genoud
Process group: SW development engineering **Process**: Software design				**Name**: Detailed design

WORK DESCRIPTION

The goal of the software design activity is to establish a software design that effectively accommodates the software requirements. During card No. 20 'Preliminary design', software requirements have been transformed into software architecture and a top-level design for the external and internal interfaces. Now, the purpose of the 'Detailed design' is to:

- Transform the top-level design into a detailed design for each software component. Components are refined into lower software units that can be coded, compiled, and tested.
- Establish traceability between the software requirements and the software designs.

…

The expected result will be materialized with a Software Design Document (SDD) in accordance with the project baseline. This document describes the position of each software unit in the software architecture and the functional, performance, and quality characteristics which each must address. The sections related to the DBMS will be on the responsibility of the card No. 25 'Database Server analysis, design and generation'.

…

Teaching resources can be helpful in writing the SDD:

- Simplified writing guide for the software design document (TEMPO-IGQ348).
- SDD Examples: Techsas and Techdis.
- Object-Oriented Analysis and Design Using UML - Ch. 10 '

…

Products	**V.**	**Milestone**
Software Design Document (SDD)	A	8-3-2009

Start date	**End date**	**Workload**	
7-30-2009	8-3-2009	5	5

Figure 3. Example of a scene card.

3) Exemplar scenes

Our fundamental problem is to prescribe the content of apprenticeship scenes, their objectives and their outcomes and to link scenes with SE processes (and their outcomes). Our modest proposition is to build scenes from previous scenes description, called exemplar scenes. Rather than being provided with an abstract definition of the prescribed situation and its prescribed tasks, the tutor has to design scenes from previous exemplar scenes and his/her own previous scenes (from past projects). Table III shows the complete breakdown for some processes.

Although an intermediate level between process and task may exist (12207 activity), the hypothesis is made that it complicates the model - and that hypothetical activities are only presented so as to facilitate the link with the 12207.

4) Competency Assessment Model

Regarding the understanding of software processes that students are building, we were faced with 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 occurred. Now, first iteration (focused on learning 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 which students are placed. We need an assessment framework that is common to both projects and which allows apprentices to relate and cumulate experiences.

TABLE III. PROCESS BREAKDOWN AND EXEMPLAR SCENES

12207	**Process**	**Hypothetical activity**	**Scene**
Group Process 'Software Project Management'			
6.3.1, 6.3.2	Project Manage-ment	• Project tailoring • Project planning • Project progress	Response to solicitation Project plan Project plan review Weekly progress meeting Project monitoring and control
…			
Group Process 'Software Development Engineering'			
6.4.1	Require-ments Capture	• Functional requirement capture • Technical requirement capture • Document requirements • Requirements review	Retro-capture of requirements Functional requirements Technical feasibility study Document requirements Non-functional requirements Architectural feasibility study Requirements review
7.1.3 7.1.4	Software design	• Design tailoring • Architectural design • Detailed design • Database design • Design review	Maintenance tasks Retro-engineering Architectural design Database analysis and design Detailed design Design review
…			
Group Process 'Software Development Support'			
6.2.1	Methods and tools support	• Process establishment • Process improvement • Tools support	Life cycle process modelling Project process modelling Process monitoring implementation Tool usage guide
…			

Software companies use assessment of software processes for capability determination and process improvement [24]. Although we think that process assessment as defined in ISO/IEC 15504 or CMMI is beyond the reach of young engineers, we believe that a simplified Process Reference Model and a personal Process Assessment Model are required to provide a basis for the practice of software engineering. Furthermore, we think that these models may provide an initial structure of the repertoire.

We observed that our apprenticeship scenes and work placement periods mobilized a similar set of apprentices' competency. As mentioned in Section §III.A.3, each process is associated with a family of competencies constituted with a list of knowledge topics and a set of abilities or skills required to perform the process. We believe that a first step in competency assessment should be made by the engineer him/herself through a self-assessment of abilities at a maturity level. The assessment scale grows from 0 to 5; - 0 - Don' know anything; - 1 - Smog: vague idea; - 2 - Notion: has notion, a general idea but insufficient to an operational undertaken; - 3 - User: is able to perform the ability with the

help of an experienced colleague and has a first experience of its achievement; - 4 - Autonomous: is able to work autonomously; - 5 - Expert: is able to act as an expert to modify, enrich or develop the ability.

Four times a year, each young engineer is required to auto-assess his/her maturity level for each ability of the 13 competency families (as well as for each transversal competency). This periodic self-confrontation with the competency reference model is called a competency inventory and is performed while auto-analyzing the tasks performed and his/her achievement level with the abilities defined in the family.

Periodic competency inventories are stored in a Content Management System (CMS). The CMS is hierarchically structured according to the group process / process decomposition. This structure is also intended to store artefacts that may be of interest in illustrating the ability determination. When the apprentice needs to relate a task performed with a process's ability, he/she has to write an entry associated with the process and may link this entry with artefacts stored. It constitutes a rudimentary portfolio, but is sufficient for our purposes.

Our system reference models are presented in Figure 4.

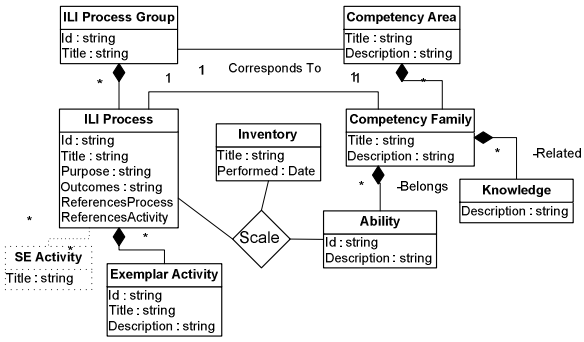

Figure 4. A model of Process Reference Model -PRM- (on the left) and Competency Reference Model (on the right). Periodic inventories hold abilities self-assessments.

5) Technical issues

As the project moves forward, information is constantly updated - in content, and in structure too. Moreover, metadata management is required. In order to support these purposes, we propose a very simple architecture based on the use of several inter-linked semantic wikis. Semantic wiki is the most flexible tool in order to record and shape a structured content.

The structural elements of these reference models do not change as projects go along and theirs events are recorded. In order to facilitate links between the project journal and these models PRM, information is stored into two semantic wikis and a Content Management System (CMS):

• 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 upper-level of the company wiki contains decompositions Processes group / Processes / Exemplar Activities and Stages / Scenes.

• a CMS contains the periodic inventories together with related artefacts used as witness of the maturity level.

B. Process enactment

1) Questioning our hypotheses

A case study, discussed below, will provide observations and information belonging to the different types discussed in §III.B (presentations, accounts and comments) and structured with the models of §IV.A (hierarchical group process/process/scenes model and competency reference model) enhanced of events' modelling of the project-in-action presented above.

Self-recording of activity is materialized by adding new items either in the portfolio or in the wiki, which is acting as a journal. A measurement of each apprentice's recording (from low to high) gives an indication that the apprentice is aware of the structure of his/her repertoire and able to use it to classify their experiences. It will provide an empirical verification of hypothesis H2: the Course-of-Action observatory may help to be aware of his/her repertoire.

Once experiences are self-recorded, apprentices are periodically performing self-assessment and self-analysis. Comparing self-assessments of previous cohorts with those of our Study Team may provide an indication that the use of an observatory is influencing the maturity level reached by the team at study.

The self-analysis of the Course-of-Action is providing a view of the enacted processes as they are reconstructed and perceived by the apprentices themselves. On the other hand, the project should follow processes as they are prescribed by the company's tutor. A qualitative evaluation of the process reconstruction gives an indication of the gap between prescribed and enacted processes. It will provide an empirical verification of hypothesis H1: self-analysis and self-assessment helps an apprentice to reveal theory-in-use.

2) An empirical case study

This case study is based on the activity of a team of 6 young software engineering apprentices, the former author as a participant-to-observe having a direct contact of the team members, sharing their environment and taking part in the activities of the team, the latter conducting formal assessments as they happen. This case study observes the whole course of the project. As pointed out by Singer and Vinson [25], apprentices' consent is required. At the beginning of the project they were informed on the field study and its objectives, and they 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 Web resources, search in different modes, browse hierarchically or with facets, and manage RDF vocabularies. The project uses Jena (http://jena.sourceforge.net) an open-source Semantic Web programmers' toolkit as RDF API.

3) Planning and monitoring the project

The project enactment is based on the process models of the previous section, a Y-shaped life cycle that separates resolution of technical issues from resolution of feature issues [26] and a typical WBS (Work Breakdown Structure: "*a deliverable-oriented hierarchical decomposition of the work to be executed by the project team to accomplish the project objectives and create the required deliverables. It organizes and defines the total scope of the project*" [27]).

The WBS has a structural and a temporal breakdown. Each process is achieved through scenes defined from exemplar activities. The WBS is temporally organized in 9 stages of 2 weeks. The planning of each stage orchestrates several work scenes. Scenes will be performed by team members, and should produce artefacts.

Information is recorded at mid-level of the company wiki (http://oysterz.univ-brest.fr/company). This mid-level structure acts as a simple but realistic model of a project: breakdown of the project stages into work scenes; allocation of persons to scenes; expected inputs and outputs. This mid-level is filled with instances (wiki pages) corresponding to the project WBS and updated regularly. The structure of this mid-level is given in the right half of Figure 5.

4) Recording the project progress

We state in §III.B.3 that software artefacts produced by the team will serve as (i) presentations. As the project progresses, events of interest are recorded in a journal associated with significant artefacts they may have used or produced. As described in III.B.3, each individual Course-of-Action is accounted for, on a 2-3 days basis, in an instance of the smaller unit, 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, artefact), and upload artefacts. At the end of each 2-week period apprentices account for 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 (ii) accounting provides a first-level of self-confrontation, as required by the Course-of-Action observatory.

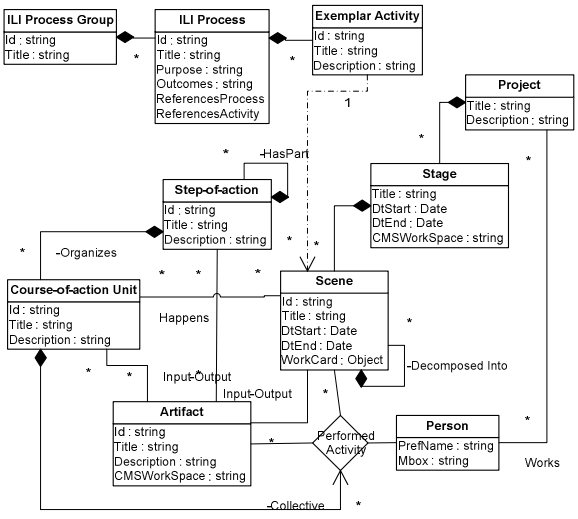

Figure 5. Representation of an enacted project. The lifecycle of a project is organized into stages, composed of scenes. During a scene, actors perform an SE activity inspired by an exemplar activity, yet contextual to the project. Input and output work products (artefacts) are linked to scene, activity and process. Self-observing the action leads to a rebuilding of project processes into steps of Course-of-Action units.

Every two months, apprentices perform a first-level Course-of-Action analysis on the state of the project's software processes. Each apprentice works on few processes and builds intermediary process elements called Steps-of-Action, based on historical Course-of-Action Units related to this process.

All information is recorded in the lower-level of the company wiki (http://oysterz.univ-brest.fr/company). The structure of this lower-level is shown on the left side of Figure 5.

5) Self-assessment

An attempt must be made to relate the university and industrial phases of the student's experience to one another. Fortunately, the competency assessment model of our system (which could be considered to be the learning objectives) is based on a simplified model of professional activities. So it may help apprentices to link up their competency building, thus avoiding their having to 'climb two ladders simultaneously' [28].

As stated at §IV.A.4, each apprentice is asked to self-analyze the activities they carried out (during both university and industrial periods) four times in the course of the year, in line with the immersion system's competency assessment model. Students assess their own maturity, on a scale of 0 to 5, for each ability or transverse competency.

In order to prepare the periodic competency assessments, apprentices use the CMS as a portfolio which hosts significant work and interesting artefacts. At any moment of the year, either in industry or at university, the apprentice may encounter a work situation, or perform a task which they perceived to be a significant experience. Within the competency reference model, they must identify one (or several) skills related to this experience, and then associate a new entry with a description of the experience, uploading artefacts that testify to this experience.

6) Building their own repertoire

The process models presented in Section IV.A are used throughout the year to structure the apprentices learning process in the reflective practicum at university (and partially in industry). The process models are also providing structure for the self-recording of apprentices' Course-of-Action and periodic self-assessments of competencies. We believe that these models provide an initial structure for the repertoire, acting as knowledge paths towards recording and retrieval practices within the repertoire.

V. EXCERPTS OF RECORDINGS AND ANALYSIS

We give some quantitative facts about the case study and empirically question the research hypotheses of Section 2.

A. Wiki accounting

The project is now complete, and Table IV gives the number of instances (wiki pages) in each category:

- 69 work scenes occurred,
- students carried out 118 Performed Activities,
- roughly 100 artefacts were produced.

For each process, we have the quantity of: Work Scenes (SCE); Performed Activities (PAY: individual); Course-of-Action Units (CAU: collective), and Steps (STE: higher-level construct). The 6th column gives an indication of the quantity of Software Engineering Activities that may be envisaged in the process. The 7th and 8th columns report the 12207 breakdown of related processes: number of activities (Act) in (the) process (es) and number of corresponding tasks (Tsk). The 9th and final column gives the number of Base Practices (BP) in the corresponding process from the 15504 standard.

TABLE IV. QUANTITATIVE FACTS FROM THE CASE STUDY.

Process	*SCE*	*PAY*	*CAU*	*STE*	*SE Act.*	*Act*	*Tsk*	*BP*
Project management	13	22	13	5	5	7	14	15
Quality insurance	2	2	1	2	2	4	16	8
Configuration management	2	2	2	3	3	6	6	10
Requirements capture	10	18	10	3	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	4	5	6	20	20
Technical support	8	16	2	2	2	3	4	6
Methods and tools support	3	3	3	3	2	-	-	6
Documen - tation	2	4	2	2	2	4	7	8
Installation - deployment	1	2	2	1	2	2	5	6

Students report on their activity at the end of each 2-week stage. Where an activity has extended beyond a single stage (e.g. technical support or coding), students adopt a simple strategy: creating one single mid-level structure (Course-of-Action Unit), and linking it to individual low-level units (Performed Activity) belonging to different stages.

B. Self-Recording

In order to evaluate the impact of the Course-of-Action observatory, we measure the use of the portfolio associated with the competency assessment model. We compare the two teams of the 2008-2009 cohort: a Control Team – which does not record its activity in an observatory – and Study Team. Each team comprises 6 apprentices.

For each process of the Process Reference Model, and for each apprentice, we measured the number of associated entries in the portfolio. Table V shows, for each team, the minimum, maximum and average number of entries.

For the Study Team, the average for each process is significantly higher than that of the other team. Remember that each apprentice of the Study Team has to report their activity in the observatory after each period at the university. Comparison of Repertoire Use Between a Control Team and the Study Team .

TABLE V. COMPARISON OF REPERTOIRE USE BETWEEN A CONTROL TEAM AND THE STUDY TEAM .

	08-09 Control Team			**08-09 Study Team**		
Process	***Min.***	***Max.***	***Avg.***	***Min.***	***Max.***	***Avg.***
Project Management	0	4	2.33	1	5	2.5
Quality Insurance	0	2	1	0	3	1.16
Configuration Management	2	3	2.5	2	5	3.5
Requirements Capture	0	7	3.83	2	8	4.66
Software Analysis	1	4	3	1	8	3.83
Technical Architecture	1	5	2.83	1	7	3.83
Software Design	3	6	4.5	3	8	5.83
Software Construction	1	7	4.16	3	8	5.33
Integration - Validation	1	4	2	1	5	3
Technical Support	2	4	2.66	2	10	3.33
Methods and Tools Support	1	4	2.16	2	8	3.83
Documentation	1	6	2.83	1	8	4
Installation - deployment	2	6	3.16	2	7	4

It is plausible to think that this periodic self-confrontation helps them to be aware of the Process Reference Model that structures the repertoire and facilitates filling the repertoire.

We may reasonably argue that our hypothesis H2 is well-grounded: the Course-of-Action observatory helps an apprentice to be more aware of the repertoire.

C. Self-assessment

A comparison of the different systems can be drawn from personal competencies follow-up. For the 13 competency families, Table V presents three self-assessment averages (in September, February and May) for the 2006-2007 cohort (previous system: no work placement), the 2007-2008 cohort (new system: with work placement), and the 2008-2009 case study team (current system: with work placement and observatory). Each cohort comprises 2 teams of 6 students.

All families make steady, and roughly equivalent, progress - with or without work placements (and with or without observatory). Due to the small number of students in cohorts, and the paucity of our statistical knowledge, no statistical comparison was performed. However, there is no evidence to indicate that the observatory helps understand software processes and reveal theories-in-use.

Some small differences can be pointed out: the 08-09 team-members assess themselves at a lower level than previous cohorts, except in terms of Project Management.

This may indicate that building the observatory, and reconstructing processes, have enforced this competency.

TABLE VI. TECHNICAL COMPETENCIES: PERSONAL FOLLOW-UP FOR THE 2006-2007, 2007-2008 COHORTS, AND 2008-2009 STUDY TEAM

	06-07 Cohort			**07-08 Cohort**			**08-09 Team**				
Competency family	**9/6**	**2/7**	**5/7**	**9/7**	**2/8**	**5/8**	**9/8**	**2/9**	***S***	**5/9**	***S***
Project Management	1.5	2.8	3.4	1.3	2.7	2.9	1.2	2.3	*1*	3.5	*3*
Quality Insurance	1.1	2.4	2.8	1.4	2.3	2.4	1	1.1	*2*	1.5	*2*
Configuration Management	1.2	1.8	2.9	1.6	2.9	3.0	1.2	1.6	*2*	2.7	*3*
Requirements Capture	2.1	3.2	3.6	1.8	2.8	3.0	2	2.3	*2*	3.2	*2*
Software Analysis	3.6	3.7	3.9	2.4	3.0	3.3	2	2.1	*2*	2.7	*3*
Technical Architecture	1.4	2.4	3.0	2.0	2.8	2.9	1.4	2.1	*2*	2.7	*3*
Software Design	2.8	3.2	3.5	2.3	3.1	3.6	1.8	2.1	*2*	3	*4*
Software construction	2.7	2.7	3.1	2.5	2.9	3.4	2.2	2.5	*2*	3	*2*
Integration - Validation	1.2	1.3	2.7	1.3	2.0	3.2	1.2	1.8	*2*	2.3	*3*
Technical support	2.3	3.0	3.4	2.4	3.1	3.5	1.4	1.9	*2*	2.3	*2*
Methods and tools support	1.7	2.6	3.2	2.0	2.5	2.9	1.2	1.8	*1*	2.5	*3*
Documen-tation	2.8	3.3	3.5	3.1	3.3	3.7	2.4	2.5	*2*	2.8	*2*
Installation - Deployment	2.4	3.3	3.5	2.9	3.3	3.7	1.6	2.6	*3*	3.2	*3*

Even though we were unable to confirm our hypothesis H1, we believe that relating the project observatory to the personal follow-up of competencies may improve apprentices' overall understanding of processes. A brief example: the complete Software Requirements Capture Process was performed in 10 scenes, spread over several sequences. Looking at the individual progress of an apprentice regarding this process, we note that her self-assessment stayed at a low maturity level of 2 - Notions - despite the fact that she had participated in several requirements-related scenes and observed her team-mates performing other related scenes. It is only after her participation in the Software Specification Requirements Document update that she assessed herself at level 4 - Autonomous – and finally perceived that the different Course-of-Action units related to requirements were related to the same field.

D. Process reconstruction

We concentrate on reconstruction by the students of higher-level Course-of-Action from the smaller units.

In Table VI, column S represents the average of the student carrying out reconstruction for this process (between February and May) - but there is no evidence that this work improved their understanding of the reconstructed process. Analysis should be correlated with the participation (and commitment) of students into scenes that are tied to the process. Further work is required.

In Table IV, the number of 12207 tasks (and 15504 Base Practices as well) give an indication as to the density of the process. The higher these numbers are, the greater the complexity - it should therefore lead to a process reconstruction involving a higher number of Steps-of-Action related to a roughly equivalent number of Software Engineering Activities. A difference between the 5th column (STE) and 6th column (SE Act.) - e.g. Requirements capture - may indicate that the reconstruction failed.

From the tutor's point of view, steps creation was haphazard. Simple processes, such as Design, have been correctly reconstructed. But, since a large number of BP (Base Practices) in Table IV indicate a complex process which may be oversimplified in the practicum (e.g. Configuration Management or V&V), the reconstruction was correct regarding the simplified process but it is partly inaccurate. For complex processes involving many scenes, reconstruction may fail - probably because apprentices are unable to perceive an abstract view of the process. This is what happened during the Software Requirement Process, where students were not able to create the Steps that would establish significant links with smaller units, nor inter-wikis links with the corresponding 12207.

VI. CONCLUSION AND FUTURE WORK

Argyris and Schön make a distinction between the two contrasting theories of action: *theories-in-use* and *espoused theories*. We proposed to adapt the Course-of-Action framework to observe software engineering apprentices' activity in the course of their final year. Two hypotheses are discussed: (1) that self-analysis and self-assessment help reveal theories-in-use, and (2) that the Course-of-Action observatory helps raise awareness of the repertoire. As a case study, the activity of a team of 6 young software engineers accompanied with two participants-to-observe is currently recorded in the observatory.

Observations are presentations (software artefacts), accounting (events in the project diary or a portfolio) and comments (steps reconstruction and activity reports). This self-observation builds a hierarchy of SE processes used as a structure for young engineers' repertoires. Four times a year, apprentices self-confront the work they did, self-assessing against a personal ability model.

Current progress with this work suggests that the process models (a personal Process Assessment Model and a simplified Process Reference Model) may form an initial structure of the repertoire, and that the observatory helps apprentices to be aware of their own experiences.

Further work is required to consider how the Course-of-Action analysis fits in with Reflection-in-Action and how it impacts the software engineering apprentices' ability to cope with innovation and change.

ACKNOWLEDGMENT

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

REFERENCES

[1] V. Ribaud, and P.Saliou, “Revealing Software Engineering Theory-in-Use through the Observation of Software Engineering Apprentices' Course-of-Action”, in Proceedings of 2009 Fourth International Multi-Conference on Computing in the Global Information Technology, New York: IEEE Press, pp. 202-210, 2009.

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

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

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

[6] 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.

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

[8] L. Pinsky, and J. Theureau, “ Activite cognitive et action dans le travail, Tome 1: les mots, l'ordinateur, l'operatrice”, Collection de Physiologie du Travail et Ergonomie, vol. 73, Paris: CNAM., 1982.

[9] J. W. Maxwell, “Using Wiki as a Multi-Mode Publishing Platform”, in Proceedings of the 25th annual ACM international conference on Design of communication, New York: ACM, pp.196-200, 2001

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

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

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

[13] 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

[14] 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.

[15] J. Theureau, “Course-of-Action analysis & Course-of-Action centered design” in: Hollnagel E. (ed.), Handbook of Cognitive Task Design, New Haven: Lawrence Erlbaum Ass., 2003

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

[17] 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

[18] J. Theureau, G. Filippi, and I. Gaillard, “From semio-logical analysis to design: the case of traffic control” in Colloquium "Work activity in the perspective of organization and design", Paris: M.S.H., 1992

[19] J. Theureau, and G. Filippi, “Analysing cooperative work in an urban traffic control room for the design of a coordination support system, chapter 4” in: Luff, P., Hindmarsh, J., Heath, C. (eds.) Workplace studies, Cambridge Univ. Press, 2000, pp. 68-91.

[20] P. Meirieu, “Si la compétence n’existait pas, il faudrait l’inventer” in IUFM de Paris Collège des CPE, 2005, (accessed April 2009) http://cpe.paris.iufm.fr/spip.php?article1150

[21] V. Ribaud, and P. Saliou, “Towards an ability model for software engineering apprenticeship”. Italics, Innovation in Teaching And Learning in Information and Computer Sciences, Vol.6 (3), July 2007, pp. 97-107.

[22] T. C. Lethbridge, S. E. Sim, and J. Singer. “Studying Software Engineers: Data Collection Techniques for Software Field Studies”, Empirical Software Engineering , vol. 10 (3), July 2005, pp. 311 – 341, doi:10.1007/s10664-005-1290-x

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

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

[25] 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

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

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

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