Revealing Software Engineering Theory-in-Use through the Observation of Software Engineering Apprentices' Course-of-action

Vincent Ribaud, Philippe Saliou

Revealing Software Engineering Theory-in-Use through the Observation of Software Engineering Apprentices' Course-of-action

Vincent Ribaud, Philippe Saliou
University of Brest, LISyC, C.S. 93837, 29238 Brest Cedex 3, France
{Vincent.Ribaud, Philippe.Saliou}@univ-brest.fr

Abstract— **Theories of action study what an actor do, in a given situation, in order to achieve consequence or objectives. Argyris and Schön made a distinction between espoused theories - those that an individual claims to follow - and theories-in-use - those that can be inferred from action -. In the software engineering field, software processes and practices constitute the espoused theory, since it is what engineers claim to follow. But what engineers - and especially apprentices - do may reveal a different theory-in-use. The capstone project provides students, working in groups, with the possibility to reflect on her/his action and that may help making explicit theories-in-use. The course of action theory considers the observable aspect of the actor's activity, i.e., what is presentable, accountable and commentable. The course-of-action observatory collects data on the courses-of-action. This observatory connects continuous observations and recordings of the agents' behavior, the provoked verbalizations of these agents in activity and the agents' comments in self confrontation with recordings of their behavior. A case study, based on the activity of a team of 6 young software engineer apprentices is used to illustrate the building and the data collecting of the course-of-action observatory and the self-reconstruction of apprentices' activity. As primary results of this work, we may think that self-observing and self-analyzing software engineer's activity help to reveal her/his theory-in-use – what governs engineers' behavior and tends to be tacit structures –and it may help them to learn more suitable theories-in-use, thus contributing to improve productivity and performance. In the special case of apprenticeship learning, it may form a part of an appropriate education intended to develop a reflective attitude.**

Keywords-component; theory-in-use; espoused theory; reflective practitioner; software engineering; course-of-action.

I. INTRODUCTION

Argyris, Putnam, and McLain Smith wrote a book on *action science* [1]. Action science is defined as an inquiry into how human beings design and implement action in relation to one another. In proposing action science, the authors hope to articulate the features of a science that can generate knowledge that is useful, valid, descriptive of the world, and informative of how we might change it [1]. Applied to software processes and practices, these concepts may help to build accurate knowledge of what they are and how they might evolve. Argyris and Schön asserted that people plan, implement and review their actions using mental maps and that it is these maps that guide people's actions rather than the theories they explicitly espouse [2]. Furthermore, they establish a distinction between the two theories of action: those theories that are implicit in what we do as practitioners and/or that can be inferred from action – the theories-in-use –and those theories that we claim to follow and on which we call to speak of (or explain) our actions to others – the espoused theories – [2]. In the software field, we observe that a software engineer may have a work behavior – her/his theories-in-use- often quite contrary to the organizations' processes, practices and procedures that s/he is supposed to follow and about s/he speaks – her/his espoused theories -.

We intended to use some action science principles 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. Excepted English and communication, no class lectures are given and all apprenticeships are project-based. For each group of 6 apprentices, a lecturer acts as a coach. As the project goes along, the lecturer gradually reduces her/his assistance in order to stand back as a participant observer. This learning system provides us with a unique occasion to instigate self-observation and inquiry of the project-in-action in order to instigate double-loop learning [2] and try to promote a reflective attitude [3].

Looking for methods to analyze, understand and improve software engineers apprentices' activities; we used the course-of-action framework in order to investigate the structural coupling of a software engineer with her/his environment and especially lifecycle software processes. The course-of-action theory, pioneered by Theureau and Pinsky [4], provides a framework for the analysis of the collective organization of the multiple courses of action in a complex, autonomous and open system. In order to make the analysis of the system dynamics, Theureau and Filippi define theoretical objects, which (1) take in account the presence of autonomous and open systems, (2) the study of which can tell how and when this autonomy and this openness can be considered as limited and (3) concerning the data it allows to consider: on the one hand, the data about the environment must include the most of which is observable, and not only the symbolic representations, and on the other hand, the data about actors must include reflexive data, and not only behavioral data [5]. As a theoretical object, matching these

three criteria, they propose to study the collective organization of multiple courses of action inside the system. Theureau and Filippi define the course of action as: "*The activity of one specific actor, actively 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 him at any moment to an observer-interlocutor* [5]." The course of action can be described from two complementary points of view: from the point of view of its global dynamics, characterizing the units of the course of action and the relations between these units; from the point of view of its local dynamics, characterizing the underlying structure of the elementary units [5]. We concentrate our analysis on the former point of view because it emphasizes on the articulation of work situations and their coordination and is more suitable to a process-level analysis. The latter point of view requires a detailed observation producing a large amount of data collected and requires a low-level analysis that is out of reach of our empirical investigation.

The central proposition of this paper is that observing and analyzing the course-of-action of software engineers – especially apprentices in a work situation - help to reveal her/his theory-in-use and facilitate learning. We begin in the next section by providing an overview of theories and systems evoked in this Introduction. In Section 3, we draft some related work. We talk briefly in Section 4 about the theories of action suitable to software engineers. In Section 5, we present excerpts of a case study. We conclude the paper with a discussion and perspectives.

II. Observing the Course of Apprentices' Project

The objective of this section is to introduce concepts and theories behind the observation system. We will present the 'sandwich' Master program and focus on the immersion system. An overview of theories of action is also provided.

A. The Sandwich Master Program 'Software Engineering by Immersion'

1) Sandwich(or work placement) courses

In France, two legal systems provide alternation of study periods in college and training periods in industry. The former system is called apprenticeship and is lasting over periods of several years (typically 3 years); it was initiated in handcraft profession but now used widely. In our local area, most companies asking for work placement students in the software field choose to use the second system called 'Contrat de professionnalisation' (professionalization contract) that spans a period of 12 months. During these 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. Strictly speaking, 'apprenticeship learning' and 'apprentice' are terms reserved to the former system, but for clarity sake, we use the term 'apprentice' in this paper.

2) Structural aspects of the program

This Master Program in Information Technology and Software Engineering is a 2-year program, accessible to Bachelor graduates in Computing or 'return to school' software practitioners. After 7-months of intensive face-to-face learning, two sub-programs are offered to students: an academic specialization in IT or a work placement system called Software Engineering by Immersion. This paper treats only of the latter sandwich sub-program and of its apprentices.

For students attending the Software Engineering by Immersion program, a requirement is to obtain a professionalization contract. Competition for this type of contract is performed during the first 7-months intensive courses. The following 4-months are dedicated to an internship period; some companies require that future work placement students have to perform their internship in the company prior the engagement, but some not. For the last year, university periods have to be intertwined with the industrial periods and we choose a two weeks / two weeks rhythm. 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.

3) Pedagogical objectives and organization

The immersion system uses a breakdown of apprenticeships into software engineering process groups subdivided into software engineering processes, together with a set of apprenticeship scenes (roughly associated with software engineering activities) which provide the learning environment and defining tasks. This hierarchical group process/process/scenes model is adapted from the ISO/IEC 12207 [6] and is used as a reference framework for the learning objectives. Table 1 in Section 4 presents the two first levels of this hierarchical breakdown and relationships with the 12207 processes. From the university point of view, this division is the reference framework in a diploma-awarding perspective. Group processes are course categories within the programme, processes are courses and scenes are classes.

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 in 9 periods of 2 consecutive weeks and work in team of 6 apprentices in order to build a complete information system. The rhythm is based on the lifecycle of a project organized into stages. Each stage was arbitrary 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.

B. Theories of Action and their Consequences on Learning

A starting point is Argyris and Schön's argument that people have mental maps with regard to how to act in situations [2]. People design action to achieve intended consequences, and monitor themselves to learn if their actions are effective [1]. Argyris and Schön made a distinction between the 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 this theory-in-use [2]." A key point in this work is the conception of human beings designing theirs actions to achieve intended consequences and that this design is governed by a set of environment variables. Argyris and Schön developed two models, called Model I and Model II, which describe features of theories-in-use that inhibit or enhance learning. Learning itself may be of two kinds: *single-loop* and *double-loop* learning. 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. This work has not been seriously deepened in this paper and further work is required to consider how course-of-action analysis is related with these organizational learning models.

C. The Courses-of-action and their Observatory

1) The course-of-action

Pinsky and Theureau, ergonomists, initiated the theoretical and methodological framework of "course-of-action", summarized in one directing idea, that of the necessity of an analysis of the actual operators' activities in real work situations for the design of new work situations [7]. An important theoretical hypothesis is that the course-of-action framework states about human activity, is that human activity is dynamically situated, i.e., always appeals to resources, individual as well as collectively shared to varied degrees, which stem from constantly changing material, social, and cultural circumstances. The course-of-action analysis add to various theories of 'situated activity' the consideration of the domain of experience, i.e., that of the agent's course-of-experience, of the constructing process of this experience at any moment, and takes an interest in the articulation between the cognitive domain and the course-of-experience. Theureau, in [8], defines the theoretical object called 'course of action' as follows: "*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 him/her at any time during its happening to an observer-interlocutor in favourable conditions*."

2) The course-of-action observatory

The course-of-action analysis is based on an observatory that allows to specify the material conditions of situated recall (time, place, material elements of the situation), the follow up and the guiding of presentations, accounts and commentaries by the agents as well as the cultural, ethical, political and contractual conditions that are favorable to observation, interlocution, and creation of a consensus between the agent and the observer-interlocutor [7]. A methodology has been developed to collect data on the courses-of-action. It connects continuous observations and recordings of the agents' behavior, the provoked verbalizations of these agents in activity (from the "thinking aloud" for the observer-interlocutor to the interruptive verbalizations at privileged moments) and the agents' comments in self confrontation with recordings of their behavior [7]. Continuous observations and recordings together with verbalizations and self-confrontation let us access to a representation of dynamics of the structural coupling between the actor and her/his situation (including other actors) [9]. A 'semiological framework' [7] provide us with a theory of activity allowing to describe the activity in abstract terms expressing hypothetical invariants. Explaining and using this theory is out of the scope of this paper focused on the observatory of course-of-action. It is sufficient to tell that this semiologic stems from the hypothesis that any period of course-of-action may be described in smaller units. The study on the course of action aims to understand its intrinsic organization and its extrinsic constraints and effects in the state of the actors, their situation and culture. This description of the intrinsic organisation of the course of action articulates two complementary descriptions: a description of its global dynamics, characterising the units of the course of action and the relations of sequencing and embedding between these units; a description of its local dynamics, characterising the underlying structure of the elementary units [6].

D. Coaching and Participant Observation

The guideline of all apprenticeships is the software development project, in which apprentices will be immersed. But a shift from apprenticeship to production must be made during this intertwined period, while the latter period of the program (from mid-May to August) will be a full-time period at the company. Hopefully, apprentices are maturing in parallel, thanks to the work placement periods, and are naturally shifting towards a professional attitude.

The project starts after the response to solicitation phase. Reference documents are project requirements and a response to solicitation. During this intertwined training course (4.5 months over a period of 8.5 months), the whole software product as defined in the requirements document is built within a framework entirely driven by the company coach. But, the coach has to gradually reduce the help offered to the student. During the first phases of the development cycle, products are assessed at the moment they are delivered, and then feedback and corrective measures are provided by the coach. In the latter phases, less feedback and help (or none at all) are given to apprentices, and they have to do the work by themselves. The coach's main tasks gradually become broad supervision, assessment and rescuing if needed. Assessment shifts from being regulation evaluation to being Validation and Verification (V&V) evaluation.

III. Related Work

Argyris and Schön's theories [2] have inseminated a large amount of research work. Organizational learning applied to the software field is the matter of the workshop series of Learning Software Organizations. In the software engineering field, Halloran [10] investigates the relationship

between a software process assessment and improvement model and organizational learning. This work points out the difference between "*engineer's espoused theory*" and her/his "*theory in use*" but it does not develop this matter as we did and rather focuses on the use of organizational learning to promote a proactive approach culturally to continuous improvement and learning procedures.

The 'course-of-action' research framework [8] consists in several empirical and technological research programs in various domains (work analysis [11], traffic control [7], sport [12], and music composition [13]). The work described in this paper uses many results of these research programs.

From the data collection techniques point of view, Lethbridge, Sim, and Singer [14] provide a useful taxonomy and an umbrella of examples from the literature. Among the techniques they define in their taxonomy, we use work diaries (§V.B), think-aloud protocols (§V.C), fly on the wall (§V.D) and participant observation (in the overall process).

In some ways, the project diary and 'sandwich reports' are similar to the progress reports and final reports as defined in [15]. But as a difference, our students must perform (and find interest in) the recording of progress reports (each 2 weeks), so it provided valuable data.

Participant observation is a used technique to collect data ([16][17]). Our goal is very similar to the aim of de Souza and Redmiles: "*to identify and analyze software developers' work practices or strategies, and by doing that inform the design of more adequate change impact analysis tools* [16]." We differ in the fact that we want to act on processes rather on tools. But as pointed out by [14], becoming too involved may lose perspective on the phenomenon being observed. In our case, the self-reconstruction of each apprentice's activity was influenced by the participant coach and it may alter the quality of observation.

IV. THEORIES OF ACTION

Theories of action study what an actor do, in a given situation, in order to achieve consequence or objectives. As noted in § 2.B, a distinction can be made between *espoused theories* - those that an individual claims to follow – and *theories-in-use* - those that can be inferred from action [2]. Espoused theory and theory-in-use may be inconsistent, and the agent may or may not be aware of any inconsistency. By definition, the agent is aware of espoused theory. Theories-in-use can be made explicit by reflecting on action [2]. In the software engineering field - and especially for young engineers or apprentices – the horizon of standards software processes and practices constitute the espoused theory, since it is what they claim to follow. But what they do (and this action is designed and do not "just happen") may reveal a different theory-in-use. We believe that making explicit theories-in-use may help software engineers – especially apprentices - to learn more suitable theories-in-use, thus contributes to improve productivity, performance and overall learning.

As the theory of course-of-action considers the part of the agent's observable activity that is pre-reflexive (i.e., presentable, accountable and commentable), without taking any interest in other aspects of the observable activity, we will obtain a simplified description - but that we estimate sufficient for the analysis - of the structural coupling of this agent and his situation. The "semiological framework" provided with the theory of course-of-action relies on three central hypotheses:

1. global dynamics - or composition: the units of courses of action are significant units for the actor (or actors) which are classified by significant structures of different ranks;
2. local dynamics - or generation: the basic unit - in other words, the lowest rank - of the course of action has the triadic sign as its underlying structure (explaining the triadic sign is out the scope of this paper, but we give a short definition: the relationship of a *representamen* - a judgment or an interpretation - to an *object* - the active engagement of the actor - through the mediation of an *interpretant* - a rule, unconscious or conscious – [11];
3. linkage between global and local dynamics - composition and generation: the transition from one unit of the course of action to another corresponds essentially to a modification in one of the three components of the triadic sign: the object.

As stated in the Introduction, we do not consider the local dynamics because it requires a too low-level of analysis and an unreachable amount of work but we are interested in reconstructing higher activities from lower units. As noted in the third hypothesis, the composition of smaller courses-of-action (e.g., a set of client's interviews) in a higher unit (e.g., a requirement elicitation activity) requires to be able to modify the *object* from the smaller units to another *object* from the higher unit (e.g., understand that performing a series of interviews is a part of requirement elicitation) and that is precisely what an apprentice or a young engineer may fail to realize. Hence, real difficulties happened while apprentices were confronted to this reconstruction and this reconstruction process worked only with apprentices having meta-cognitive abilities.

Our objective is to expose, as far as possible, information related to espoused theories, theory-in-use and course-of-action applied to our apprenticeship by immersion system. As the project goes along, information is constantly updated in content but also in structure. 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. This section will present our action models and the corresponding wikis structure.

A. Espoused theories

Software companies use software engineering and software quality standards as the foundation of their quality assurance process or of their quality management system. As software companies claim to follow and respect standards, we may think that these standards constitute a part of espoused theories of software engineers, especially Process Assessment and Process Reference Models. Process assessment such as ISO/IEC 15504 is defined as a disciplined evaluation of an organizational unit's processes against a Process Assessment Model [18]. A Process Assessment Model is a model suitable for the purpose of assessing process capability, based on one or more Process

Reference Models [18]. A Process Reference Model (PRM) is a model comprising definitions of processes in a life cycle described in terms of process purpose and outcomes, together with an architecture describing the relationships between the processes [18].

1) Process Reference Model

Our process reference model is adapted and simplified from ISO/IEC 12207; we are using 13 processes organized with 3 process groups, together with a set of apprenticeship scenes which provide the learning environment and defining tasks. Table 1 presents the two first levels of this hierarchical process group/process/scenes breakdown and 12207 links.

TABLE I. PROCESS BREAKDOWN

Process group	***Process***	***12207:2008 related processes***
Software Project Management	Project management	6.3.1, 6.3.2
	Quality insurance	7.2.2
	Configuration management	7.2.3
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 structural elements of this Process Reference Model (PRM) do not change as projects go along and theirs events are recorded. In order to facilitate links between the project journal and this PRM, information are stored into a dedicated application called eCompas and two semantic wikis:

- the 12207 wiki [19] is a hypertext reference of the ISO/IEC 12207:2008 for the process level: title, purpose, list of outcomes and process decomposition in activities and tasks;
- the upper-level of the company wiki [20]. This semantic wiki is used for many purposes and divided in several levels. The upper-level contains the structure and information about group process / process / exemplar activities.

The structure of the two wikis is given in the left side of Figure 1.

2) Competency Assessment Model

While apprentices are currently learning by doing software processes, process assessment will not measure a capability level but, in the best case, a learning capability level. Because apprentices are building competencies and that some reflective learning is required, we choose to promote self-assessment of personal abilities. In 2006, we carefully analyzed the whole apprenticeship scenes of each process in order to establish 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* [21]") that theses 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 an example for the Requirement Capture Process in Table 2).

TABLE II. AN EXAMPLE OF A COMPETENCY FAMILY: SOFTWARE REQUIREMENT CAPTURE.

Knowledge topics	***Abilities and skills***
* 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.	To mobilize specification methods and tools in a real project : • within an ISO 9001-like baseline, • in relation with requirements traceability, • to produce a Software Requirement Specification

So we kept the 2-level decomposition 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 placed abilities and transversal competencies. The whole decomposition (3 areas, 13 families, 48 abilities and 11 transversal competencies) is called an ability model [22].

The ability model represented on the right side of Figure 1 is recorded in the eCompas tool [22].

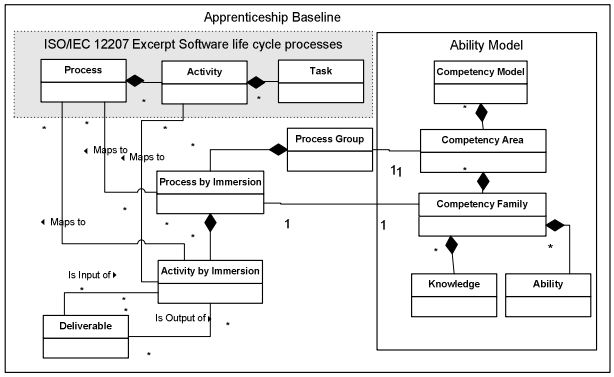

Figure 1. The structure of our Process Reference Model.

B. Theories-in-use

We defined an apprenticeship/production framework called ILI (Ingénierie du Logiciel par Immersion, Software Engineering by Immersion), based on the Process Reference Model of the previous section, a development cycle and a typical WBS (Working 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. The WBS organizes and defines the total scope of the project* [23]").

We use a Y-shaped life cycle that separates resolution of technical issues from resolution of feature issues [24]. First, the cycle is divided into two branches (tracks): a functional track and a technical track. Then these two tracks amalgamate for the realization of the system.

The WBS has a structural and a temporal decomposition. Each process is structurally decomposed in Software Engineering activities (to distinguish it from the activities in

the 12207 sense) that may have slightly variation from a project to another. Each Software Engineering activity is further decomposed in sub-activities that can be fully specified or just named, depending of the scope and goals of the project. 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.

All information is recorded in the mid-level of the company wiki [20]. This mid-level structure acts as a simple but realistic model of a project: breakdown of the project stages into work scenes; membership of project and allocation of persons to scenes; expected inputs and outputs. Each coach of a 6-student team (called a company) fills this structure with instances (wiki pages) corresponding to her/his project WBS and has to update it regularly. The structure of this mid-level is given in the Figure 2.

C. Course-of-action observatory

1) An Observatory of Software Apprentices' Activity

Recall the definition of the course-of-action in §II.C: what, in the observable activity of an agent […] is pre-reflexive or again significant to this agent, i.e., (i) presentable, (ii) accountable and (iii) commentable by him/her at any time during its happening […]. Software workers do not achieve complex technical gestures or do not have to progress along a detailed procedure. So (i) presentations to an observer are quite difficult to reproduce and presentable artifacts that are most notable and representative of the job are the outputs of software activities and tasks. (ii) Accounts are easier to collect and observe because a minimum of traceability and reporting is performed in any organization and if it is not sufficient, accounting can be provoked without significantly modify the course of the activities. (iii) Comments are not natural objects and have to be provoked: reports, self competency assessment (§ V.C).

2) What is recorded in the observatory?

Products and documentary resources are main objects of (i) presentation as they describe the inputs and outputs of the activity. The 'historical' context of resources' use and products' production has to be recorded too. This can be described in terms of events and processes, involving occurrences of agents (people) and artifacts (products and resources) meeting in space (in case of distributed cooperation) and time. As a first stage, we may consider individual courses of action of the various participants. At a second level, a collective action involves parts of several individual courses of action which take place synchronically or sequentially. We need to divide individual course-of-action in smaller units, that we call course-of-action unit. Each event of interest has to be (ii) accounted in an instance of Course-of-action Unit in relation with people and artifacts involved. It provides a kind of project diary or journal. A journal may be seen as a kind of reflective practice that is a device for working with events and experiences in order to write (iii) comments and extract meaning from them. All information is recorded in the lower-level of the company wiki [20]. Figure 2 shows the structure of middle and lower levels of this semantic wiki.

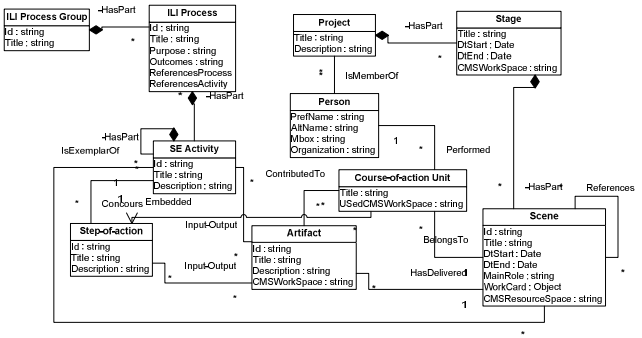

Figure 2. The structure of the observatory of apprentices' activity.

3) Self-assessment

An attempt has to be made to relate the university and industrial phases of the student's experience. Fortunately, the ability model of our system (that could be considered as the pedagogical objectives) is based on a simplified model of professional activities. So it may ease apprentices to link their competency building and avoid that apprentices were 'climbing two ladders simultaneously' and that ascent up the university ladder was unrelated to progress on the other in their firms [25]. Our ability model establishes a structure that directly supports the personal and team construction process of the knowledge and skills required to practice engineering of a software project. For each ability or transverse competency, the student assesses himself/herself at a maturity level. The assessment scale grows from 1 to 5; - 1 - Smog: vague idea (or even no idea at all); - 2 - Notion: has a 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.

It is not easy for a student plunged into the 'doing' to keep in mind the abilities aimed at by the apprenticeship scene or the work scene and to establish links between all kinds of learning. That is the reason why, at four key moments of the year, each apprentice is asked to self-analyze the activities s/he did (during university and industrial periods) with regards to the immersion system's ability model. So, four times during the year, apprentices have to establish this inventory and communicate it to their coach. This periodic inventory is supported by eCompas, a tool intended to manage development, assessment and value-added of competencies over the course of a curriculum or a professional career. The eCompas tool is intended to store artifacts that may be interesting to illustrate the ability determination. Each time a software engineer self-assesses a process's ability level, s/he has to write an entry associated with the process and may link this entry with artifacts stored. It constitutes a rudimentary portfolio, but sufficient for our purposes.

D. Scalability and generalization of immersion

The education given in the software engineering apprenticeship by the immersion system relies on the following principles:

- being centered on the competencies to be developed and on fruitful apprenticeship situations, rather than contenting ourselves with the teaching of knowledge, subject by subject;
- developing an active and co-operative pedagogy based on the project and the role play: students' immersion in a several months project which imitates a project in a firm as closely as is possible;
- working in a team, communicating about the work done, co-operating with colleagues.

These principles should apply to different curricula, provided that they are closely related to a profession (e.g., software engineer). Hence, the immersion approach should theoretically apply in other domains. But it should be noted that it took two years before the immersion system may be considered stable and operational and that most of the work was related to define the content of apprentice scenes (what competencies are addressed and how these competencies may be developed) in comparison to the organizational aspects. Unfortunately, this amount of work is not reusable and has to be established for each different curriculum.

The approach is clearly not scalable. The first reason is that it requires a new project each year in order to guarantee successful conditions (complexity, unknown, uncertainty, hazard indeed) for this constructivist approach. Another reason is related to the integrated approach: depending on the situation, the coach may try to develop any competency from the ability model and it does not ensure that all competencies may be developed in a same manner at the end of the project. Moreover, although considerably reduced regarding the 12207 standard, the envisaged area embraces many lifecycle processes and a team member may not participate to some processes and thus miss to develop corresponding abilities.

V. EXCERPTS OF A CASE STUDY

We will illustrate previous sections with a case study.

A. The Project

This case study is based on the activity of a team of 6 young software engineering apprentices with the two authors acting as observers: the former 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 depicts some aspects of the building and the filling of the course-of-action observatory. The project itself, a semantic annotation tool, is depicted in Figure 3.

As related in the previous section, the whole observatory is supported with several semantic wikis, and a dedicated application. Semantic wiki is the most flexible tool in order to record and shape a structured content. Properties (modifying the underlying data model) can be added, updated or deleted as the project goes along. Information (data) can be recorded in a bulk mode and the typesetting performed later. Things to do or to report are created in one Wiki word to indicate that they have to be filled. Information can be temporary missing or incomplete.

Functions - 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.

Technical environment - The system uses a three-tier architecture in which the user interface, functional process logic, computer data storage and data access are developed and maintained as independent modules, on separate platforms. Sub-systems are: Oracle database, Hibernate persistent layer, Spring framework running on Tomcat, JSF for the user-layer.

Documentation – The project starts from the statement of work with expected needs. Main deliverables provided by the team members are: Meeting report, Project Plan, Requirement Specification, Software Analysis, Software Design, Code, Integration and Validation Plan, Software User Manual, and Software Operator Manual.

Besides these engineering documents, team-mates produce other kinds of document related to specific needs: case study, usage guide, evaluation report, best practices, etc.

Figure 3. The semantic annotation tool project.

B. Presentations and accounts

Semantic wikis offer a lightweight authoring plate-form and will be used to record most events of the day-to-day life in the project journal. The company's coach (in this case study, the former author) initially fills and updates the WBS of her/his project in the project journal. Team members can record events as they happen but have to systematically fill the wiki at the end of each stage (2 weeks).

"*Requirement capture for a first work package*"
* Study of the domain: semantic web, RDF and RDFS language.
* Set-up of semantic wikis in order to familiarize the team with the domain.
* Study of the statement of work in order to gather requirements.
* Retro-engineering of semantic wiki technology in order to establish and write use cases.

"*Tutorial for SPARQL*"
* Building integrated queries on the test site and writing of a tutorial.
* Installation of the front-end ARQ on a Linux platform and writing of an installation guide.
* Building SPARQL queries with ARQ and writing of a tutorial.

"*Software specification requirements document update*"
* Update of all mock-up taking in account the designers' remarks
* Update of all use case packages
* Renumbering requirements
* Re-reading the document
* Typesetting and misprints correction

Figure 4. Excerpts from Course-of-action units issued from scenes related to the Software Requirements Capture Process

Recall that we define in §IV.C that software artifacts produced by the team will serve as (i) presentations. As apprentices fill their stage progress report, they have to create a Course-of-action unit (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. This (i) accounting provide a first-level of 'verbalizations' (in a written form) and self-

confrontation as required by the course-of-action observatory (cf. §II.C).

This project has completed its 9 stages and apprentices created 29 Course-of-action units related to 90 pages describing an individual performed activity. When looking at these pages, it should be noted that their content is mainly descriptive (see some excerpts in Figure 4). We found only one exception where a student recorded that he did not well organize his work, but this student knew that he failed the demanded activity and probably wishes to prevent criticisms. So we may conclude that these records in the project journals are (ii) accounts but not (iii) comments.

C. Comments

So as noted in §IV.C, (iii) comments are not a natural object and have to be provoked in a manner related to the objectives of the analysis: assessment, improvement, competency development …

The immersion system use a kind of assessments fully integrated with the life cycle, constituted by the reader/author feedback cycle, progress meeting and peer reviews. We call it regulation assessment referring to De Ketele "*[…] an open process whose priority function is to improve the working order […] of a part or of the whole system* [26]." We record artifacts produced by regulation assessments: lecture notes, progress meeting report, peer review reports which constitute valuable inputs for further analysis.

Periodic inventories of team members are recorded within the eCompas tool. A copy (in a Word format) is stored into the observatory. Focusing on the Software Requirements Capture Process, the complete process was performed in 7 scenes and 10 individual units about performed activity were recorded in the project diary (a description of 3 units is given in Figure 4). Looking at the individual advance of an apprentice regarding this process, we may note that she has participated to 3 scenes (over 7). As the year started, she assesses herself at the maturity level 1 – Smog - for the process as a whole and for each associated abilities. Inside her industrial position, she acts as a software developer and has very little opportunity to improve requirement skills. After the second periodic inventory (performed in mid-January), she assesses herself to a maturity level of 2 - Notions - despite the fact that she had participated to two scenes related to requirements and sees her team mates performing other related scenes. It is only after her third and last activity on this subject (the Software specification requirements document update depicted in Figure 4) that she assesses herself at the level 4 - Autonomous – and perceived that these different course-of-action units were related to the same field. This constitutes an example of (iii) useful comment for the apprentices but also for the coach acting as project manager in order to assign tasks to team members.

D. Analysis

The next step in the course-of-action framework is to perform an analysis. The analysis of this data produces a decomposition of the global dynamic in terms of smaller units and the relations of sequencing and embedding between these units. As noted at the beginning of §IV, we concentrate on the reconstruction by the apprentices of course-of-action steps of higher ranks from the smaller units. In the case of the Software Requirement Process, this reconstruction failed: apprentices created a single Step-in-action and a single Course-of-action unit embedding individual performed activities but were unable to establish neither significant links between units nor inter-wikis links with the corresponding 12207 Processes. A better case occurs with the Software Design Process but the reconstruction was performed by apprentices having conducted the Software Design Process Review, a formal review that helps them to understand the process at higher level.

E. Reconstructing the coach's activity by him/herself

As pointed out by Argyris and al., "*Theories-in-use can be made explicit by reflecting on action. But we should note that the act of reflection is itself governed by theories-in-use. Becoming an action scientist involves learning to reflect on reflection-in-action, making explicit the theories-in-use that inform it, and learning to design and produce new theories-in-use for reflection and action* [1]." This occurs when, preparing this paper, the former author was looking what happened into the Software Requirements Process for his six past projects. Firstly, he noted that 6 years ago, he used only three scenes for the overall process, and secondly he observed that since 3 years, he was using 6 or 7 scenes, depending of the advance of students, and he become aware of a learning pattern that he wants that the students follow.

VI. Conclusion and Future Work

We proposed to adapt the course-of-action framework to software engineering apprentices' activity in the course of their final capstone project. An observatory collects the data necessary to study the course of action therefore including continuous observations of the behavior of action and communication in a work situation as well as different kinds of instigated verbalizations (transcript in a written form) from the actors which would provide access to other elements such as interpretations, feelings, judgments. 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. As units of courses of action are significant units for the actor, we choose to breakdown the whole course-of-action in units based on individual performed activities.

We begin to use these data to proceed with the analysis of course-of-action, using a theoretical framework, described as semio-logical. This framework will make possible to explain the global dynamics - or composition - of the courses of action units, their local dynamics - or generation - and the linkage between these two dynamics.

The current state of this work let suggest that analysis will lead (1) to specify the modalities of engineers' interaction with life cycle processes leading to the design of better interaction or of help situations and (2) to contradict or support the reconstruction by the engineer of her/his own

activity, i.e., going from 'pre-reflective consciousness' of the actor towards a reflective attitude.

Thus, we may think that observing and analyzing software engineer's activity help to reveal her/his theory-in-use [2] i.e., what governs engineers' behavior and tends to be tacit structures – that we may call Project Processes-in-use because the project processes constitutes the engineer's reference rather than a Process Reference Model. However, the unit breakdown of course-of-action is based on performed activities related to a simple Process Reference Model issued from the ISO/IEC 12207:2008 standard. We made the hypothesis that this standard constitutes the "*espoused theory*" of most software engineers. So, the course-of-action framework may help engineers to establish a link between her/his "*Project Processes-in-use*" and "*espoused Process Reference Model*" and contribute to reduce the fit between a project-in-action and simplified SE standards.

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 [27] 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. Further work is required to consider how course-of-action analysis is related with these organizational learning models and hence, on the software engineers (and apprentices) ability to cope with innovations and changes.

As another perspective, we may consider a Personal Assessment Model, based on our analysis of software engineer's abilities and incorporating a simplified version of the exemplary Process Assessment Model of 15505:2004 standards – especially Base Practices [18] – which may reduce the fit between "*Process Assessment-in-use*" and "*espoused Process Assessment*".

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, 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 and D. Schön, "Theory in practice: Increasing professional effectiveness", San Fransisco: Jossey-Bass, 1974, pp. 6-7.

[3] D. Schön, "The Reflective Practitioner", New York: Basic Books, 1983.

[4] 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.

[5] J. Theureau and G. Filippi, "Metropolitan traffic control activities and design of a support system for the coordination of actions in future control rooms", Proceedings of the Fourth European Meeting on Cognitive Science Approaches to Process Control (CSAPC '93), Copenhagen, 1993.

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

[7] 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.

[8] 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

[9] F. Varela, "Principles of biological autonomy", New York: Elsevier, 1980.

[10] 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.

[11] 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

[12] J. Theureau, "Selfconfrontation interview as a component of an empirical and technological research programme" in : II° Journées internationales des sciences du sport, Paris, 2002.

[13] N. Donin and J. Theureau, "Music composition in the wild: from the horizon of creative cognition to the time & situation of inquiry" in: EACE 05, Crète, pp. 57-64, 2005.

[14] 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

[15] F. Shull, F. Lanubile, and V. R. Basili, "Investigating reading techniques for object-oriented framework learning" IEEE Transactions on Software Engineering, Vol. 26 (11), Nov 2000, pp. 1101- 1118, doi:10.1109/32.881720

[16] C. R. B. de Souza and D. F. Redmiles, "An empirical study of software developers' management of dependencies and changes", in : Proceedings of the 30th international conference on Software engineering (ICSE '08), New York: ACM, pp. 241-250, 2008

[17] C. Seaman and V.R. Basili, "Communication and organization: an empirical study of discussion in inspection meetings", IEEE Transactions on Software Engineering, Vol. 24 (7), Jul 1998, pp. 559-572, doi:10.1109/32.708569

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

[19] 12207 wiki, http://oysterz.univ-brest.fr/12207 May 29, 2009

[20] Company's wiki, http://oysterz.univ-brest.fr/company May 29, 2009

[21] P. Meirieu, "Si la compétence n'existait pas, il faudrait l'inventer" in IUFM de Paris Collège des CPE, 2005, http://cpe.paris.iufm.fr/spip.php?article1150 May 29, 2009

[22] 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.

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

[24] P. Roques and F. Vallée, "UML en action", Paris: Eyrolles, 2002.

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

[26] J.M. De Ketele and X. Roegiers, "Méthodologie du recueil d'informations", Bruxelles : De Boeck, 1993.

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