Towards an abitlity model for software engineering apprenticeship

Vincent Ribaud, Philippe Saliou

TOWARDS AN ABILITY MODEL FOR SOFTWARE ENGINEERING APPRENTICESHIP

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

ABSTRACT

Despite recent efforts to improve the effectiveness of software engineering education, most approaches do not equip students with non-technical skills and fail to be practice-oriented. Brest University provides the software engineering by immersion paradigm as an alternative to other education systems. Shifting to the constructivism paradigm as far as possible, this education system is entirely based on a 7-months project, performed by a 6-students team within a virtual company and tutored by an experimented software engineer.

The ISO/IEC 12207 standard is a reference framework of software engineering processes. This standard provides the basis of our reference decomposition into processes/activities/tasks. Issued from professional didactics, the analysis of activity distinguishes two kind of activity: productive and constructive. The former is work-oriented while the latter helps the actor to improve his/her own practice hence is apprenticeship and personal development. Analysing apprenticeship scenes provides an ability model of our immersion system. The model is defined in terms of its constituent competencies areas, each of which is further defined in terms of its constituent competencies families; a family corresponding to an activity of the reference decomposition. Each family is associated with a set of cohesive abilities.

The 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. Each student periodically fills this structure while auto-analysing the tasks performed and him/her achievement level with the abilities defined in the model. 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.

Keywords
Software engineering, apprenticeship, reflective practices, abilities management.

1. INTRODUCTION

Recognizing a core body of knowledge is pivotal to the development and accreditation of university curricula and the licensing and certification of professionals. For example, the Guide to the Software Engineering Body of Knowledge project is an initiative from IEEE to identify and describe a subset of the body of knowledge that is *generally accepted* within the profession of software engineering (IEEE Computer Society, 2004), ”generally accepted” meaning established traditional practices recommended by many organizations. On the other hand, most software engineering professionals will say that they learned the profession by doing. Hence a paradigm used is teaching software engineering by doing. Most academic curricula address this issue through projects, but academic projects are not sufficient to achieve the goal. Several analyses of software engineering teaching emphasise the benefit of a long-term team project (one semester or a year) (IEEE and ACM, 2001), (Pour and al., 2000), (Meyer, 2001).

As Donald Schön (1983) pointed out, technical rationality treats professional competence as the application of privileged knowledge to instrumental problem of practice. But to take up the challenges of their practice, the practitioners place one’s reliance on their repertoire of experience and certain ingeniousness gained during their practice rather on formulae learned during their fundamental education. Difficulties of an initial education in software engineering are to establish the balance between knowledge and skills and to provide an initial structure to an experience repertoire.

Since September 2002, the second year of Brest University Master in software engineering is entirely based on a 7-months project, performed by 6-students team. The main objective is mastering software engineering activities and skills. Moreover, among the characteristics and practices recommended for computer science

graduates (Computing Curricula 2001, Computer Science Volume), we kept as additional objectives: working in a team, coping with change, and being able to appreciate a perspective in a whole.

The pedagogy used to achieve these objectives breaks with the usual teaching paradigm to rely on an apprenticeship paradigm (Tardif, 1998). This pedagogy is achieved at university through the immersion within a virtual company, without traditional courses. Students work in team to analyse, design, implement, test and document a management information system, following the software engineering rules of the art and tutored by an experimented software engineer (Saliou and Ribaud, 2003). The immersion system belongs to the constructivism approach, which can be summed up with two fundamental statements: learning is defined as an active process for knowledge building rather than a knowledge acquisition process; teaching is essentially aimed at helping students in this process rather than transmitting knowledge (Duffy and al., 1996).

The reference framework of our education system relies on a breakdown of the software engineer profession into activity areas (called processes) and activities within processes. Learning each activity is performed within several apprenticeship scenes (often linked upstream and downstream with others scenes). For each scene, a card describes objectives, resources, tasks to perform, expected results … Looking at our education system from the competency point of view, the 2-level breakdown applies and structures an ability model into competencies areas and families. Each family further breaks down into abilities. A set of knowledge is globally associated with each family.

Professional didactics analyses practitioners' activity. As an apprentice, we learn by doing but also by analysing what we did and how we did it. Since September 2006, our students experiment the auto-analysis of their activity with regards to the ability model of the education system by immersion. The goal of this experiment is to initialize a personal follow-up of competencies that could be pursued in a professional career. It is intended to help the student to focus on his/her evolution and on the development dynamics of his/her competencies. Managing competencies requires to set up an initial model and to periodically perform an auto-assessment of current acquisitions. For this purpose, we developed *eCompas*, a Web-based information system intended to help students, faculty and academic administrators to manage development, assessment and value-added of competencies over the course of curriculum.

2. SOFTWARE ENGINEERING EDUCATION

2.1 Software engineering book of knowledge and curriculae

Software engineering can be minimally defined as the set of activities involved in developing, operating and maintaining software. Software development is organized around a project which involves people guided by a software development process. The process organizes the set of activities allowing to transform users' requirements into software products. Products are artifacts (or deliverables) that are created during the life of the project, such as specifications, architectural design, source code and documentation.

Thus, the discipline of software engineering can be seen as an engineering field with a stronger connection to its underlying computer science discipline than the more traditional engineering fields. The definition of the body of the Software Engineering Education Knowledge (SEEK) by IEEE and ACM reflects the reliance of software engineering on computer science (IEEE and ACM, 2004).

The Guide to the Software Engineering Body of Knowledge (SWEBOK) is a comprehensive description of the knowledge needed for the practice of software engineering. One of the objectives of this project was to "provide a foundation for curriculum development and individual certification and licensing material" (IEEE Computer Society, 2004). The product of the SWEBOK project is not the body of knowledge itself, but rather a guide to it. This Guide seeks to identify and describe that subset of the body of knowledge that is generally accepted by many organizations.

The Guide to the SWEBOK as well as the SEEK uses a two- or three-level hierarchical organization. The highest level of the hierarchy is the knowledge area, representing a particular sub-discipline of software engineering. Knowledge areas (KAs) are high-level structural elements used for organizing, classifying, and describing software engineering knowledge. Each area is broken down into smaller divisions called units in the SEEK and topics in the Guide, which represent individual theme within an area. Each unit is further subdivided into a set of topics (called sub- topics in the Guide), which are the lowest level of the hierarchy.

The structure of the Guide is widely used as a basis for curriculae decomposition. Bagert and Mu (2005) reviewed 43 software engineering Master's degree programs in the United States. 8 of the 10 KAs of the Guide to the SWEBOK - Requirements, Design (including Architecture), Construction, Testing, Engineering (Project) Management, Process (including Measurement/Metrics), Tools and Methods, Quality (including Verification and Validation) - are covered by more than the half of the programs. Only 2 KAs - Maintenance, Configuration Management- are marginal. Moreover, most of programs have chosen to take a "depth-first"

approach to the various software engineering KAs (breaking down the discipline in areas and topics) instead of the “breadth-first” paradigm of having one or more general survey courses.

2.2 Difficulties of software engineering education

Despite the efforts undertaken to harmonize the structure and contents of software engineering education, some criticisms exist. Shaw (2001) argues that current approaches do not equip students and learners with essential non-technical alongside technical skills and knowledge (e.g. team working and communication skills). A most damaging (and old) criticism argues that the current concept-oriented curriculum is well-suited for preparing research engineers, but not the practice-oriented engineer on which competition increasingly depends (Denning, 1992).

A common misconception about software engineering is that it is primarily about process-oriented activities (i.e., requirements, design, quality assurance, process improvement, and project management). In this view, competency in software engineering can be achieved by acquiring a strong engineering background, a familiarity with a software development process and a minimal computing background, including experience using one or more programming languages (IEEE and ACM, 2004). Such a background is, in fact, quite insufficient: software engineering is computing too. While it is not expected that every software engineer will have deep expertise in all of aspects of computing, a general understanding of their relevance and some expertise in particular aspects are a necessity (IEEE and ACM, 2004).

Furthermore, teaching (and learning) software engineering from an engineering perspective is extremely difficult. Software engineering courses are regarded by students as abstract, academic and quickly forgotten concepts. Without experiencing their practical impact on realistic programs, students rarely develop a deep understanding or appreciation of important ideas in software engineering.

2.3 Software engineering by immersion

Industry complains that graduates take at least one year to become productive once hired. So, the main idea of our education system is to let professional realities into our university walls. Students work in teams to analyse, design, implement, test and document a software project relying on strong software engineering principles. The pedagogical system imitates as closely as possible real-world phenomena: a professional working environment, the client-supplier relationship, the application of a development baseline, the use of methods and associated tools, the cooperation within the team. We call this education system «software engineering apprenticeship by immersion» (Saliou and Ribaud, 2003).

Let us present immersion principles. Students do not follow any courses, excepted English and communication skills. Teams of students are immersed in a realistic project going on in the university and tutored by an ex-professional engineer. The project is the global context to learn software engineering skills. The whole apprenticeship plan of action is guided by a staged development process, under the control of the corporate baseline of Thales, a software services company. A different technological framework and tools set are provided to each team. In order to emulate a standard firm environment, we fitted out laboratory rooms with a landscape room for each team, common meeting rooms and machine rooms. Each student has his/her own individual work station.

The immersion system provides a reflective practicum as D. Schön (1987) defined it. The nature of educative experiences of the apprenticeship by immersion system is quite different of other education systems.

> […] the experience of the students in any reflective practicum is that they must plunge into the doing, and try to educate themselves before they know what it is they’re trying to learn. (Schön, 1987).

Software engineering planning and development process should result in a definition of what products are to be produced, who is to produce them, and when they are to be produced. The whole apprenticeship plan of action is guided by the 2-TUP (2-Track Unified Process) development cycle (from requirements to deployment), which defines, among others, the role and the schedule of project stages. The apprenticeship process is elaborated and tailored according to learning goals but also to real work progress and team’s specificities. It reveals clearly three kinds of teaching activities:

- Organising. It is a matter of scheduling and elaborating work cards that define, at each stage, the work to be done and preparing pedagogical supplies (book, software, corporate baseline, real examples …) needed to carry out the work.
- Tutoring. For each work card, a tutor is available to students who are thus provided with continuous support and assistance.
- Control. Work cards constitute the assessment framework because they define form and content for the deliverables to be produced and delivered.

The year is divided in three periods, called iterations. During the first 4-month iteration, students are swapped around the tasks dictated by engineering activities. The tutor's help and guidance happen on a daily basis. During the second 2-month iteration, roles are fixed within each team and the companies are relatively autonomous during the completion of the project, the tutor performing mainly a supervision and rescue activity. Finally, the third iteration, the training period, is in fact an operational period.

3. FROM AN ACTIVITY MODEL TOWARDS AN ABILITY MODEL

As stated before, a recognized profession must have an established body of knowledge and skills that its practitioners understand and use consistently. On the other hand, typical software engineering usages and skills are gathered in corporate baseline which defines good practices and capitalizes the company's know-how but diffusion and use are restricted to the company.

Software engineering standardization is an attempt to integrate, regulate and optimize existing best practices and theories. For example, the International Standard ISO/IEC 12207 aims to establish a common framework that can be used by software practitioners to manage and engineer software. The standard establishes a top-level architecture of the life cycle of software. The architecture is built with a set of processes and interrelationships among these processes. Each process is further designed in terms of its own constituent activities, each of which is further designed in terms of its constituent tasks. This standard is a reference framework of processes and activities of software engineering.

The activity analysis gives the opportunity to clarify some aspects of competencies at work. How can we identify and describe competencies in order to act on their development, at work or as being apprenticed. The immersion system simulates software engineering activities for the purpose of apprenticeship. The apprenticeship situations reproduce the reality but the richness of situations is preferred to the reliability of reality reproduction. The apprenticeship model inherits from a part of the activity model but also presents an organisation stemming from learning: the apprenticeship scenes. We learn a lot in action but we learn so much with the analysis of our action. We have to accompany the learner's activity while setting up some reflection procedures on his/her activity. It can be done while describing his/her experiences and apprenticeships, and thinking about gained or missing knowledge and skills in order to move towards a development of his/her competencies.

Meirieu (2005) defines a competency as « the ability of a person to act in a pertinent way in a given situation in order to achieve specific purposes". Competencies are means of action which a person has to perform his/her productive activity with regards to the situation he/she has to deal with. The immersion system aims to acquire professional competencies that we prefer call abilities. We built an ability model for our education system, structured with areas, families, abilities. The model is a reference to regularly set up an achievement level for each ability or transverse competency. This regular set-up helps student to be aware of competencies objectives and to realize the advance towards these objectives.

This section aims to present the different conceptual models (processes and activities model in § 3.1 ; professional didactics in § 3.3) which underline the operational models of the education system (apprenticeship model in § 3.2 ; ability model in § 3.4).

3.1 The ISO/IEC 12207 processes and activities model

ISO/IEC 12207 (1995) is the first international standard to provide a comprehensive set of life cycle processes, activities and tasks for software that is part of a larger system, stand alone software product, and software services. Amendment 1 (2002) et 2 (2004) provides an revision to ISO/IEC 12207 that establishes a co-ordinated set of software process information that can be used for process definition and process assessment and improvement. This standard constitutes a world-wide agreement on what activities make up a software project.

ISO/IEC JTC 1 N7846 (2005) recall some definitions of ISO/IEC 12207. Processes are described at a summary level by the use of Title, Purpose, and Outcomes. The process attributes are as follows:

- The title conveys the scope of the process as a whole.
- The purpose describes the goal of performing the process.
- The outcomes express the observable results expected from the successful performance of the process.
- The activities are a list of actions that may be used to achieve the outcomes.

Rather than describing the results of executing a process, activities describe a possible set of actions that might be undertaken to execute the process. According to ISO/IEC 12207 clauses:

"Each process is further defined in terms of its own constituent activities, each of which is further defined in terms of its constituent tasks. An activity within a process is a set of cohesive tasks."

"A task is expressed in the form of a requirement, self-declaration, recommendation, or permissible action."

The set of activities should, when performed, assure the achievement of the process outcomes. ISO/IEC JTC 1 N7846 (2005) states that unlike the process/activity relationship, the set of tasks within an activity is not required to "cover" the activity. We interpret the notion of task in a different manner: the tasks of an activity provide a structural decomposition of the activity and “cover” the activity.

3.2 The apprenticeship model

3.2.1 Reference framework

The upper-level of ISO/IEC 12207 reference model provides the upper-level of our reference framework. Our apprenticeship by immersion system structured these activities around three main processes: software project management activities, specific software development activities and software development support activities. From the 25 processes of ISO/IEC 12207, we concentrate on those related to software development cycle, that is: 5.3 Development, 6.1 Documentation, 6.2 Configuration Management, 6.3 Quality Assurance, 6.4 Verification, 6.5 Validation, 7.1 Management, 7.2 Infrastructure. The ISO/IEC 12204 Amendment 1 proposes a grouping of processes into categories. We reorganized the selected processes above in a same manner. The main process considered is the Development process. The other considered processes are not so wide than the development process, so there are retrograded to activities and reorganized in two processes: Project Management and Development Support.

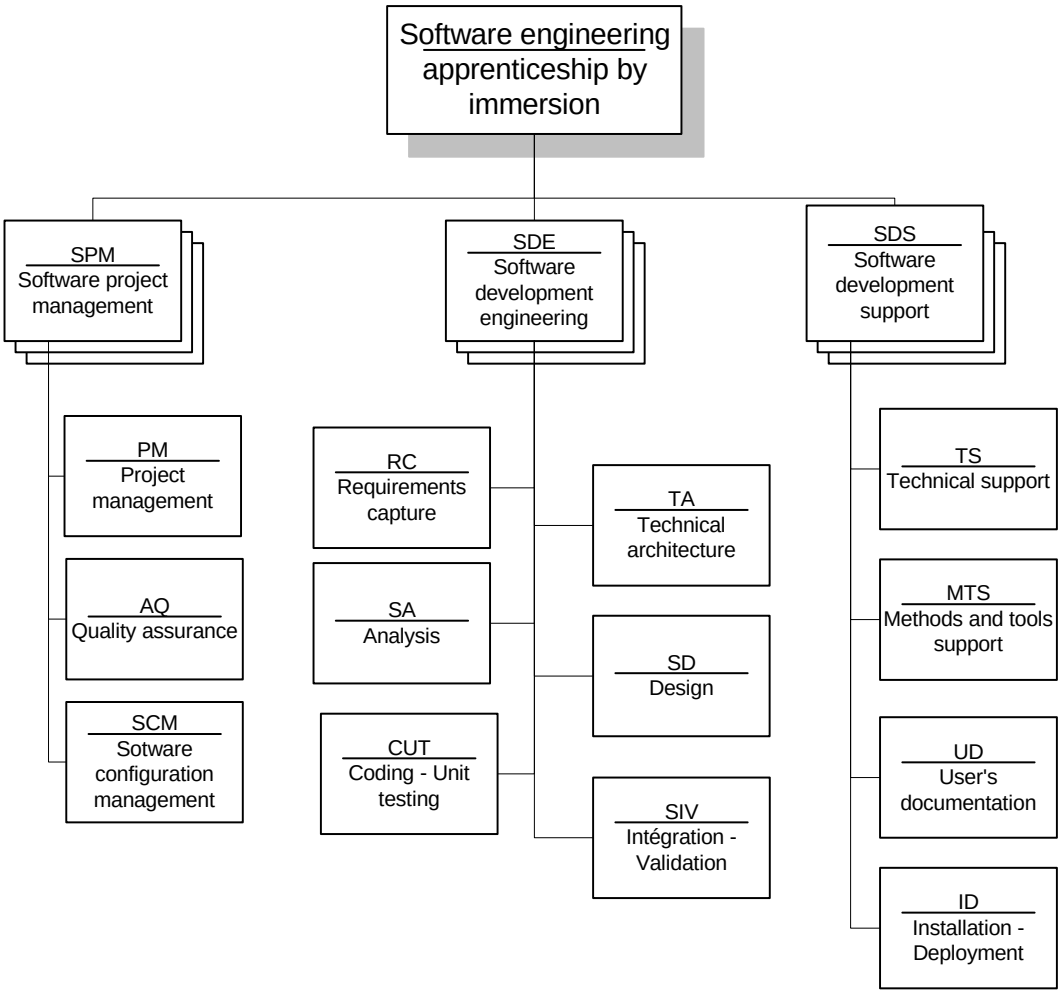


Figure 1: Activities break-down

In the ISO/IEC 12207, the Development process consists of the following activities : System requirements analysis and system design; Software requirements analysis; Software design; Software construction; Software integration; Software qualification testing; System integration and qualification testing. Our

decomposition slightly differs because we do not need system activities (only software) and we put emphasize on computing constraints, essentially Technical Architecture.

Each activity represented in figure 1 can be analysed from different points of view: required knowledge and skills ; stakeholders' roles ; input and output deliverables ; required tools and resources …

At work, what makes sense for these multiple point of views is their articulation within the activity situation (the cohesion of the work situation). At apprenticeship, it is also the (apprenticeship) situation which let to understand the multi-dimensional nature of activity. The system by immersion is a theatre play where the different actors learn to play their roles. The apprenticeship scene is the reference context where a part of the play happens: the apprenticeship scene aims at a unity of place, time and action; the scene is together a situation where students learn and do, an scenario of actions, a role distribution, an area mobilizing resources and means. The different components of a scene along with their articulation are depicted in a card, called apprenticeship card (cf. § 3.2.3).

There are two reference decompositions that meets exactly for the two first levels ; the former is an activity model coming from the ISO/IEC 12204: process, activity, task ; the latter is an apprenticeship model: process, activity, scene. A scene is related to a single activity but aims to learn several tasks of the activity. Conversely, a task can be undertaken within several scenes of the same activity.

3.2.2 Apprenticeship process

Several scenes happen simultaneously. During the same period of time, students work in subgroups on different activities belonging to different processes. Furthermore, learning objectives of an activity could require the cut out in different scenes closely coupled. The complete cycle of scenes is temporally organised into stages. Each stage (from 1 to 3 weeks) groups several scenes and carries on activities belonging to the three processes. The cycle is: Stage 0 : Introduction (1.5 week) ; Stage 1 : Project and means set-up (1 week) ; Stage 2 : Requirement capture - Architecture choices (2.5 weeks) ; Stage 3 : Requirements consolidation (2.5 weeks) ; Stage 4 : Analysis - Technical exploration (3 weeks) ; Stage 5 : Design tailoring (1.5 weeks) ; Stage 6 : Design (1.5 week) ; Stage 7 : Realisation - Integration (2.5 weeks) ; Stage 8 : Validation (1 week) ; Stage 9 : Deployment - Verification (1 week).

Each scene of a stage is described with an apprenticeship card that define precisely the work to be done along with input and output products and the apprenticeships to gain and/or the competencies to mobilize. Each apprenticeship card is assigned to one or several students who should take up the job (or the role) inherent to the activity. Students are swapped around the roles from one stage to another stage. For each apprenticeship card, students can rely on tutor's support and assistance on a daily basis.
Each apprenticeship card gives rise to one or several deliverables. Each deliverable is carefully examined and annotated by tutors, and then the tutor feeds back comments to the authors together with improvements to bring about. This assessment and feedback process is iterated (at least twice) until that the deliverable is considered as good enough for its future exploitation (it should arise some problems when the final delivery is not judged as good enough).

Recall that the immersion system relies on two apprenticeship cycles called iterations. At the end of the first iteration, students carried out (or saw their team mate at work) different engineering activities and students used deliverables produced by other students during previous stages. We consider that they have a first "apprenticeship by doing" of the software engineering activities (cf. figure 1). A fixed organisation of the team is set up for the second iteration, structured around roles. One student is acting as project manager while the others are carrying out all the others activities. During the second iteration, students will rely on the apprenticeship process acquired during the first iteration. The team has to transform this process in a production process. They generally reuse most of apprenticeship cards as work cards.

3.2.3 Apprenticeship scenes

The apprenticeship process strongly relies on apprenticeship cards describing the scenes; then students enrich them for their own software development (productive) process. The card structure is standardized (see an example below). Its main elements are the process/activity (here development/design) tied up to the work; the role to play (here designer or architect) with students' name; 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 anlysis and design course); workload and lead-time information.

Number : 24	**Date** : 12-13-2006	**Origin** : Ph. SALIOU	**Roles assignment**	
Program : SIGILI3g	**APPRENTICESHIP CARD**		Designer Architect	**A. Guenard** **R. Lucas**
Process: Software development engineering **Activity** : 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 a previous apprenticeship card N°20 "Préliminary 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. The components are refined into lower levels containing 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 TEMPO-ILI 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 apprenticeship card N° 25 "Database Server analysis, design and generation".

…

Pedagogical resources can be helpful in order to write the SDD :

- Simplified writing guide for the software design document (TEMPO-IGQ348).
- SDD Examples : Techsas and Techdis.
- Oracle course "Object-Oriented Analysis and Design Using UML" - Chapter 10 "System Architecture"

…

Products	**Version**	**Milestone**
Software Design Document (SDD)	A	12-22-2006

WORK IN PROGRESS

Estimation					**Reality**				
Start date	**End date**	**Workload**			**Start date**	**End date**	**Used workload**		
12-13-2006	12-22-2006	5	5						
Date	**Used workload**	**Deliveries-Observations**							

Figure 2: Apprenticeship card

3.3 The analysis of activities and competencies

Apprenticeship by immersion belongs to a practices field that is professional education where the theoretical framework of professional didactics applies. Pastré (2004) characterizes professional didactics with two viewpoints:

- It looks learning from the activity point of view: learning in a professional situation is not knowledge assimilation but is learning to act in a durably efficient way.
- It looks learning from the human subject's development point of view, precisely his/her competencies development: we do not focus on the work situation and how the subject deals with it but we put emphasize on the actor in term of his/her future and of his/her competencies development dynamics.

Professional didactics questions in priority practitioners' activity. Learning by the activity (by doing) has always been a kind of learning. Samurçay and Rabardel (1995) distinguish two faces in human activities. The *productive activity* is an activity made, oriented and controlled by the human subject to perform tasks he/she has to achieve with regards to the situation features. The *constructive activity* is oriented and controlled by the

subject that performs it in order to build and develop competencies with regards to the situation and professional areas of action. *Competencies* constitute the proper resources that the subject disposes to perform his/her productive activity; competencies are mobilized as means in the work activity.

The apprenticeship system by immersion is a big role play which a part looks like life-size simulation. Within the professional education area, simulators and simulation gave rise to an important literature which can be there a source of inspiration. Patrick (1992) states that « the simulation has to present the task to the learner in such a way that psychological or competencies requirements are not significantly changed. The learner has to perform the same cognitive activities while doing the simulated task and the reference task." The immersion system still reduces the distance between simulated task (that should be rather called approximated task) and the reference task. Situation-problems present in the work activity are presented in apprenticeship by immersion situations, called scenes (cf. § 3.2). The activity engaged in a scene has two sides. It is a productive activity because tasks (even approximated) are realized. It is also a constructive activity, which is a learning activity, where learning is achieved while performing the task. The activity analysis let us clarify the competencies question and their evolving dynamics.

It is not easy for a student plunged into the doing to keep in mind the abilities aimed by the apprenticeship scene (and by the education system) and to establish links between all kinds of learning. It is not easy too to be aware of the relationship between productive and constructive activities. That is the reason why since September 2006, at 4 key moments of the year, each student is asked to auto-analyse the activities with regards of an ability model of the immersion system. Gathering the analysis is facilitated with the use of eCompas, a tool that we developed for this purpose.

3.4 Towards an ability model

3.4.1 An ability model

Although there are relationships between Knowledge Areas and topics of a topic-oriented curriculum like the SWEBOK and activities of a software engineering framework like the ISO/IEC 12207, links are not easy to establish. As we started in 2002, we addressed this problem while structuring the software engineering field around main processes such as project management or development and while decomposing each processes into activities such as assurance quality or requirement capture. Knowledge Areas refers either to processes or activities of the framework and topics of the KAs take place where the reference matches up. In 2006, we carefully analysed the whole apprenticeship scenes of each activity in order to establish the abilities that theses scenes are intended to develop. We tried to answer to the question "what is the student able to do, once the scene performed?". This analysis gave us a set of abilities for each activity. So we kept the 2-level decomposition of our reference framework, the first level being called competency areas (corresponding to processes) and the second level competency families (corresponding to activities), and we placed abilities within each activity. We call the whole decomposition an ability model of our education system.

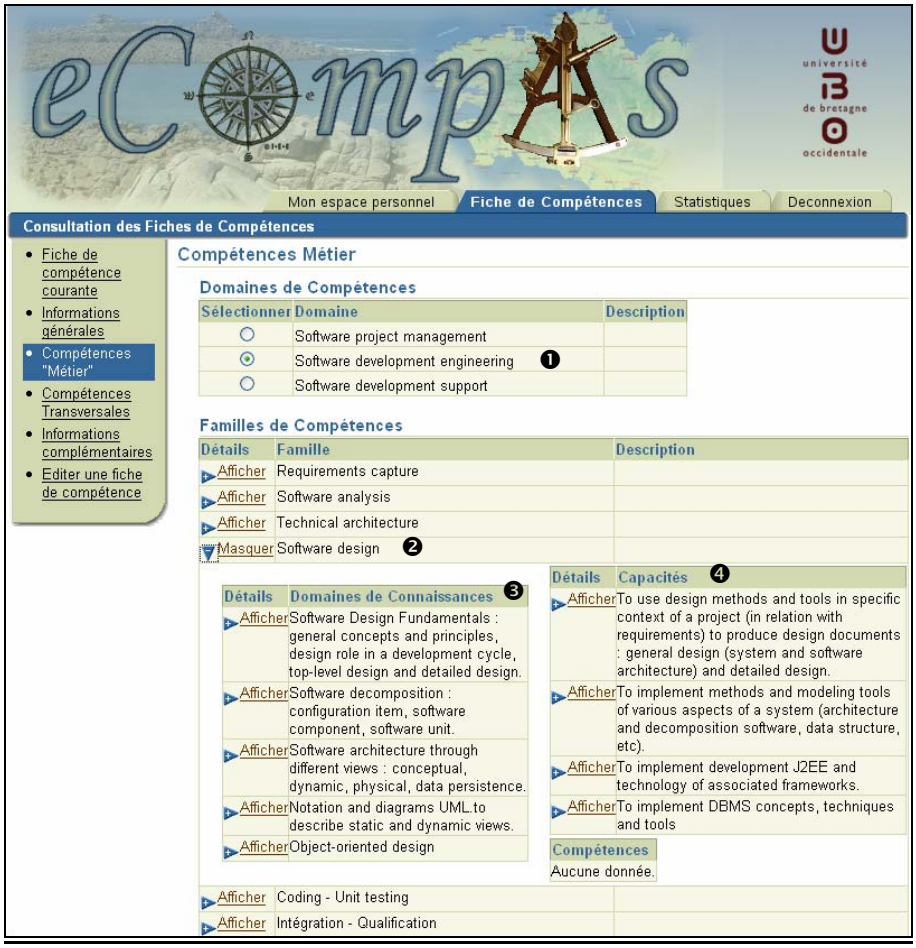
eCompas is a tool initially intended to manage the ability model of our system. During the design of the tool, we decided to implement a data model and functions able to support several ability models related to different curriculae or professional organizations. Then, eCompas is a general-purpose competencies management tool. Main entities used and implemented in the eCompas tool are defined as follows.

A *competencies model* (or competencies ontology) defines and contains the data decomposition. A competencies model is used to define the elements of a curriculum (or a professional baseline) and their relationships. During the education period, the whole students use the same model but can individually change afterwards.

A model is decomposed into *competencies areas***;** each area roughly corresponding to one of the main division of the profession or of a curriculum. For example, a model used for employees of a system integration department could contain the areas: system integration, validation & verification, configuration management.

Each area organizes the competencies into *families*. A family roughly corresponds to main activities of the area. Thus, configuration management's role requires competencies that should be organized into families: configuration management process implementation ; configuration identification and management ; configuration management tool ; configuration audit ; release management and delivery.

Each family is made of a set of *knowledge* and *abilities,* eventually *competencies*; each of these entities being represented by a designation and a detailed description.



This screen snapshot illustrates a subset of the abilities model used in the system. Once selected the competencies area ❶ and then the competencies family ❷ (« Design » in our case), the system displays descriptive information in terms of aimed knowledge ❸ and abilities❹.

3.4.2 Personal follow-up of competencies

Applied to the software engineering apprenticeship field, the ability model (3 processes, 13 activities, 48 abilities and 11 transversal competencies) 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. When students realize this structure of the SE field, it provides them with an initial (but empty) repertoire of experience. The filling of their repertoire can be helped through the periodic confrontation of what they did with regard to the ability model. Four times during the year, students have to establish this inventory called competencies compendium and communicate it to their tutor. We call this periodic inventory the Personal Follow-up of Competencies (PFoC) and this activity is supported by the eCompas tool.

Ideally, each time a stage is completed; students examine the tasks performed, why they acted as they did, what was happening in order to be aware of what knowledge and skills they learned. Recording these observations is provided through a log feature for each ability. For each ability or transverse competency, the student assesses himself/herself a maturing level. The assessment scale grows from 1 to 5; - 1 - *Smog*: vague idea (even no idea at all); - 2 - *Notion*: has notion, a general idea but insufficient to an operational undertaken; - 3 - *User*: is able to perform the ability with the help pf an experienced colleague and has a first experience of its realization; - 4 - *Autonomous*: is able to work autonomously; - 5 - *Expert*: is able to act as an expert to modify, enrich or develop the ability.

Because of its professional roots, the software engineering apprenticeship by immersion curriculum is skill-oriented. The first step for students is to realize that this orientation differs from the previous knowledge-oriented curriculae. This awareness happens after the first time they evaluated their achievement level with regards of the ability model.

3.4.3 The eCompas tool functions

The eCompas system was created to help students, faculty and academic administrators to manage development, assessment and value-added of competencies over the course of curriculum. Four main functions have been defined each of which can be helped by functionalities of eCompas: those functions are competency model management, personal follow-up of competencies (PFoC), situation analysis of competencies, institutional supervision.

The first function aimed by eCompas is the management of competencies models. Definition and organisation of competencies within families and areas is the starting point of a model that is available in eCompas. This model can then be used to identify competencies, skills and knowledge as well as different levels of achievement that can be reached through formation and related with different areas of formation. This can be then used as a chart to identify which families and competencies has to be gathered to design a curriculum or a formation path

The second function is the personal follow-up of competencies development (PFoC). With the help of competencies models and charts of achievement levels available on eCompas, students will be able (with minimal training or additional assessment tools) to assess their own level of achievement regarding specific competencies identified on the system. It will then be possible to record on eCompas this « snapshot » on several occasions and be in a position to report on the progression of competencies development related both to professional experiences and to curriculum progression.

The third function is basically aimed at producing situation analysis of competencies. eCompas technical functionalities permits to record and to use models of competencies as well as snapshots of achievement levels in order to help “return to school advisor” to evaluate on what terms mature students coming from professional background can integrate a specific curriculum.

The fourth and last function aimed at by eCompas functionalities is institutional supervision. Faculty using e-compass will then be in a position to produce, on a cohort basis, comparable statistics about competencies development and assessment as well as progression between two compendiums on specific chosen points of curriculum progression.

4. CONCLUSION

The interest of an activity model (process/activity/task) of a profession is to supply a referent framework for doing the job. Our immersion system exploits this reference decomposition in order to organize learning with apprenticeship scenes, each scene “setting to music” different tasks of software engineering. The analysis of activity performed during the scenes provided an ability model that coincides with the activity model on the two-first levels; each activity being associated with a cohesive set of abilities.

The immersion system works successful since 5 years. Since September 2006, we experiment the use of the eCompas tool, intended to help students, faculty and academic administrators to manage development, assessment and value-added of competencies over the course of curriculum. Four times a year, students establish a compendium through an auto-assessment (on a scale from 1 - Smog - to 5 - Expert -) of their maturity for each ability. It is intended to accompany the activity with awareness and assessment on competencies growing. This usage could profitably be pursued during a professional career, thus helping reflective practice and personal development.

5. ACKNOWLEDGEMENT

The authors wish to thank the anonymous referees for the constructive and helpful comments that led to restructure the manuscript and improve the final result.

6. REFERENCES

Bagert, D. J. and Mu X. (2005), Current State of Software Engineering Master’s Degree Programs In the United States, In *Proceedings 35th Frontiers in Education*, Indianapolis.

Denning, P. J. (2002), Educating a New Engineer, *Communications of the ACM*, December 1992.

Duffy, T. M. and Cunningham, D. J. (1996) Constructivism: Implications for the design and delivery of instruction, In *Handbook of Research for Educational Communications and Technology*, MacMillan.

IEEE and ACM (2001), Computing Curricula, Computer Science Volume, http://www.sigcse.org/cc2001 (last accessed May 6th, 2007).

IEEE and ACM (2004), Software Engineering 2004, Curriculum Guidelines for Undergraduate Programs in Software Engineering, http://sites.computer.org/ccse (last accessed May 6th, 2007).

IEEE Computer Society (2004), Guide to the Software Engineering Body of Knowledge, Overview http://www.swebok.org/overview/ (last accessed March 26th, 2007).

ISO/IEC 12207:1995 (1995), Information technology -- Software life cycle processes, International Organization for Standardization (ISO), Geneva, Switzerland.

ISO/IEC 12207:1995/AMD 1:2002, International Organization for Standardization (ISO), Geneva, Switzerland.

ISO/IEC 12207:2002/AMD 2:2004, International Organization for Standardization (ISO), Geneva, Switzerland.

ISO/IEC JTC 1 N7846 (2005) SC 7 Request for New Work Item Proposal – System and Software Engineering – Life Cycle Management - Guidelines for Process Definition, ISO/IEC JTC 1, New York.

Meirieu P. (2005), Si la compétence n'existait pas, il faudrait l'inventer, In IUFM de Paris Collège des CPE, http://cpe.paris.iufm.fr/spip.php?article1150 (last accessed May 20th, 2007).

Meyer B. (2001), Software Engineering in the Academy, *IEEE Computer*, May 2001.

Pastré P., (2004), Introduction In *Recherche en didactique professionnelle*, edited by Samurçay, R. and Rabardel, P., Octarès, Toulouse (France).

Patrick, J. (1992), *Training, Research and Practice*, Academic Press, London.

Pour G., Griss M. L. and Lutz M. (2000), The Push to Make Software Engineering Respectable, *IEEE Computer*, May 2000.

Ribaud, V. Saliou, P. (2003), Software Engineering Apprenticeship by Immersion, in *International Workshop on Patterns in Teaching Software Development, ECOOP'03*, Darmstadt (Germany).

Samurçay, R., Rabardel, P. (1995). Work competencies: some reflections for a constructivist theoretical framework. In *Proceedings 2nd Work Process Knowledge Meeting: Theoretical approaches of competences at work,* Courcelle sur Yvette (France).

Schön, D. (1987), *Educating the Reflective Practitioner: Toward a New Design for Teaching and Learning In the Professions*, Jossey-Bass, San Fransisco.

Schön, D. (1983), *The reflective practitioner*, Basic Books, New York.

Shaw, M. (2000), Software Engineering Education: a Roadmap, In *Proceedings of the Conference on The Future of Software Engineering*, Limerick (Ireland).

Tardif J. (1998), Intégrer les nouvelles technologies de l'information – Quel cadre pédagogique ?, ESF, Thiron (France).