A few elements in software development engineering education

Vincent Ribaud, Philippe Saliou

A few elements in software development engineering education

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

Abstract

Brest University offers the software engineering by immersion paradigm as an alternative to other education systems. The idea is that students follow through a project from A to Z, relying on an ISO9001 quality management system alongside methods and tools associated with present n-tier architecture - but under apprenticeship conditions. Software engineering activities are structured around three main processes: Development Engineering, Project Management, and Development Support. Focussing on Development Engineering, we report on certain challenges and difficulties, illustrated on a real-scale project.

1. Introduction

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 inside our university walls. Except for English and communication courses, no lectures are given. Students work in teams to analyze, design, implement, test and document a software project relying on strong software engineering principles. The educational system imitates real-world phenomena as closely as possible: a professional working environment, the client-supplier relationship, the application of a development baseline, the use of methods and associated tools, and cooperation within the team. We call this education system 'Software engineering apprenticeship by immersion' [6]. The main goal of the system is to educate skilled but also reflective practitioners, as defined by D. Schön [8]. Focussing on Development Engineering, we report on certain challenges and difficulties, illustrated on a real-scale project.

2. Elements of the Software Engineering Programme

2.1. Structure of the programme

Our programme leads to the Software Engineering Master's Degree. This short paper discusses only the final year of the programme. We use a hierarchical process/activity/scenes model (adapted from the ISO/IEC 12207 [3]) as a reference framework for this final year. Software engineering activities are structured around three main processes: Development Engineering, Project Management and Development Support. From the university point of view, this division is the reference framework in a diploma-awarding perspective. Processes are course categories within the programme, activities are courses and scenes are classes.

2.2. Objectives of the Development Process Category

We use a decomposition issuing from the Development process in the ISO/IEC 12207 (without the system-level activities): software requirements analysis; software design; software construction; software integration; software qualification testing.

The concerns of each course are not described here but can be found in the SEEK [2] or the SWEBOK [10]. We reproduce our educational objectives of the Development Process [12]:

- Requirements: to use specification methods and associated tools within the context of an ISO 9001 quality management system; to manage requirements.
- Analysis: to perform a domain analysis and a requirement analysis intended to define detailed requirements and the business data model.
- Design: to use design methods and design and modelling tools, linked with requirement management, in order to produce expected deliverables: general (architectural) design and detailed design.
- Construction: to transform the detailed design into object code; to help debugging and testing activities through programmeming discipline and the use of tools.
- Integration – Qualification: to design and write An Integration and Qualification Plan; to master an integration framework and its associated tools; to write qualification tests.

3. A few elements for discussion

3.1. Project objectives and constraints

We took one of the projects (called KnowKnow) carried out this year as an example.

The main goal of KnowKnow is to provide a semantic annotation tool able to annotate (indexing through metadata), 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).

RDF is a W3C standard intended to manage metadata. RDF models metadata as 3-tuples which assert that some resource (identified by URI) has some property (identified by URI) which has a value identified - either by URI or given literally. We use Jena - a leading Semantic Web programmers' toolkit - as RDF API. Jena is an open-source project, implemented in Java, and available for download from SourceForge [4].

The KnowKnow 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.

3.2. Some challenges

Legacy and complexity. A major challenge for students is to deal with the complexity of legacy systems. As the new project starts, one part of the team has to work on the project carried out the previous year with the (little) help of documentation written by past students. Students have to re-install the system, put it into service, set up the development environment and begin fixing some existing defects.

The KnowKnow system is intended to replace a semantic extension of the Sakai Collaboration and Learning Environment. Semantic means the ability to create, retrieve, query and navigate with knowledge about the entities managed by Sakai. RDF is the support of this semantic extension, through the integration of Jena within Sakai. The semantic extension as well as the Sakai environment itself is a huge bazaar of technologies, tools, ways of doing and so on.

This legacy environment is a part of the reflective practicum and an essential element of learning. "[…] 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. The teachers cannot tell them" [8].

Adaptability. Another challenge is to be able to adapt his/her way of doing to the actual needs. No single software development process exists that is universally applicable. A major difficulty for students is the specialization of a (theoretical) reference process so that it fits its situation.

On a real-project, tailoring of the development process generally falls to the project manager. This requires time and continuous effort. Tailoring practices refer to a process; they are intended to improve the process, but their main purpose is to contribute to the process realization, i.e. to achieve the process goals and deliver required outputs. In our system, students perform tailoring practices in order both to deal with technological and methodological unknowns and enhance team performances.

On the KnowKnow project, students performed several tailoring activities:

- requirements: defining use cases format and dealing with the case of CRUD use case
- analysis [respectively (resp.) design]: writing a usage guide from the analysis activity (resp. design activity) as it is proposed in the Unified Process, then retro-engineering the design (resp. the code) of a project performed last year in order to produce an analysis document (resp. a design document) for this project
- tests: retro-engineering a Test Plan of a real industrial project in order to understand the what and the how of a Validation Plan.

Design. D. Parnas introduces his seminal paper as "This paper discusses modularization as a mechanism for improving the flexibility and comprehensibility of a system while allowing the shortening of its development time" [5]. This seems obvious, but it was only after performing a large design and its afferent coding phase that our final year Masters' students discovered that subsystem decomposition is essentially aimed at splitting and assigning the development of subsystems to different team members, thereby allowing parallel development of individual subsystems.

So, software design is a critical task - hard to teach and hard to learn. The studio is the central training method in architecture schools and this analogy was used to provide a suitable educational environment for software design. "Students are encouraged to do the work while being self-aware of the decisions they make and of the actions they take […] Coaches basically maintain a vision of best practices, observe the student's efforts to match that vision, and give advice to help them in later attempts" [11].

Design activity is strongly related to the mastery by students of a complex development environment. On the KnowKnow project it took from one to two full months of work for each student (performed within different apprenticeship scenes) to deal with this technical challenge and to be able to understand the technical issues of the design.

Design is also very strongly related to requirements and tests. Students have to allocate requirements to the software components they are designing and demonstrate traceability over the whole cycle, from validation to requirements. On the KnowKnow project, most students find this harder to achieve than the technical issues.

3.3. Some difficulties

Several attempts to adopt the studio for software engineering education have been made at different universities. Experiences differ, however, and some papers report

very satisfactory results. However adopting studios may raise difficulties either from within the faculty organization itself, or by revealing resistance from both faculty members and students.

Ideally, lessons learned in the studio should be used in traditional courses but this requires closer interaction between courses and teachers' co-operation. This is one of the reasons why we built an immersion learning environment, but it requires that tutors be experienced software managers.

Perhaps the hardest thing to manage is the success of the projects. Depending on the variables of student motivation and skills, tutor performance, project interest, and other unknowns, the progress of a studio can be very close to what was expected or radically different - ranking from approximation to disaster. From our own experience, a failing project is a difficult experience for both students and teachers; it may be difficult to find lessons in the failed studio and tutors may be deterred from repeating the experience.

4. Conclusion

Shaw [9] identifies several challenges and aspirations as targets of progress for software engineering education: identifying distinct roles in software development and providing appropriate education for each; instilling an engineering attitude in educational programmes; keeping education current in the face of rapid change, and establishing credentials which accurately represent ability.

Our answer tries to combine learning by immersion environment - intended to offer to students a reflective practicum - with a well-structured learning process which aims to organize productive and constructive (learning) activities. This paper has discussed just a few of the many educational challenges and difficulties.

5. References

[1] ACM and IEEE, Computing Curricula 2001, Computer Science Volume, chapter 11, http://www.sigcse.org/cc2001/cs-graduates.html (last accessed March 1st, 2008).

[2] ACM and IEEE, Software Engineering 2004, http://sites.computer.org/ccse (last accessed March 6th, 2008).

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

[4] Jena – A Semantic Web Framework for Java, http://jena.sourceforge.net/ (last accessed March 6th, 2008)

[5] D. Parnas, "On the criteria to be used in decomposing systems into modules", Communications of the ACM Volume 15 Issue 12 , ACM Press, New York, 1972, pp. 1053 - 1058.

[6] V. Ribaud, and Ph. Saliou, "Software Engineering Apprenticeship by Immersion", International Workshop on Patterns in Teaching Software Development, ECOOP 2003, LNCS Volume 3013/2004 ,Germany, 2003, p. 137.

[7] Sakai home page, http://sakaiproject.org/ (last accessed March 6th, 2008).

[8] D. Schön, "Educating the Reflective Practitioner", Meeting of the American Educational Research Association, Washington D. C., 1987.

[9] M. Shaw, "Software Engineering Education: a Roadmap", In Proceedings of the Conference on the Future of Software Engineering, Limerick (Ireland), 2000, pp. 371-380.

[10] IEEE Computer Society (2004), Guide to the Software Engineering Body of Knowledge, Overview http://www.swebok.org/overview/ (last accessed March 6th, 2008).

[11] J. E. Tomayko, "Carnegie Mellon's software development studio: a five year retrospective", In Proceedings of the 9th Conference on Software Engineering Education, IEEE Computer Society Press, pp. 119-129.

[12] (in French) Université de Brest, Master informatique, Ingénierie de Développement Logiciel, http://dept-info.univ-brest.fr/pages/masque-centre-iup.htm (last accessed March 6th, 2008).