Bootstrapping an empty repertoire of experience: the design case

Philippe Saliou, Vincent Ribaud

Bootstrapping an empty repertoire of experience: the design case

Philippe Saliou

Complex System Laboratory, LISyC
University of Brest, C.S. 93837
29238 Brest Cedex 3, France
Philippe.Saliou@univ-brest.fr

Vincent Ribaud

Complex System Laboratory, LISyC
University of Brest, C.S. 93837
29238 Brest Cedex 3, France
Vincent.Ribaud@univ-brest.fr

Abstract

Performing good design is a difficult task. To take up this challenge, practitioners rely on their repertoire of experience. Students, however, do not have any such repertoire. We propose an approach aimed at bootstrapping the repertoire. The approach is generally accomplished in two steps: tailoring the activity – acquiring a minimal structure through a deductive approach, then initializing the repertoire through an inductive approach; and performing the activity - to begin filling the repertoire whilst drawing up the design of a real-scale project.

Categories and Subject Descriptors D.2.2 [**Software Engineering**]: Design Tools and Techniques – Modules and interfaces.

General Terms Design.

Keywords software engineering, reflective practitioner, design

1. Introduction

A core body of knowledge is generally seen as pivotal to the development and accreditation of university curricula and the licensing and certification of professionals. Both the Software Engineering Body of Knowledge (SWEBOK, [1]) and the Software Engineering Education Knowledge (SEEK, [2]) use a two- or three-level hierarchical organization: knowledge area, topics (SWEBOK) / units (SEEK), sub-topics (SWEBOK) / topics (SEEK).

Donald Schön [3] pointed out several problems with this science-based model of professional preparation. "*The situations of practice are not problems to be solved but problematic situations characterized by uncertainty, disorder, and indeterminacy*" (pp. 15-16); "*situations of practice are characterized by unique events*" (p. 16). Hence, to take up the challenges of their practice, practitioners rely on their repertoire of experience, together with a certain ingeniousness gained during their practice, rather than on knowledge-oriented curricula or formulae learned during their basic education. D. Schön defines the repertoire as: "*The practitioner has built up a repertoire of ideas, examples, situations and actions. […] A practitioner's repertoire includes the whole of his experience insofar as it is accessible to him [her] for understanding and action*" [3, p. 138]. A problem to be solved can be analyzed from different points of view: expected knowledge and skills; stakeholders' roles; input and output deliverables; required tools and resources. At work, what makes sense for these multiple points of view is their articulation within the activity situation (the cohesion of the work situation). We think that the structure of a practitioner's repertoire is activity-oriented rather than knowledge-oriented. Practitioners generally use a hierarchical model of professional activities. In the SE field, we assume that a Process Reference Model (such as the ISO/IEC 12207 process/activity/tasks model) may be used as the reference framework of a software engineer's repertoire.

Students, however, do not have any such repertoire. The difficulties of an initial education in software engineering lie in establishing the balance between knowledge and skills, and in providing an initial structure to an experience repertoire. Hence one paradigm in use is the teaching of software engineering 'by doing'. Most academic curricula address this issue through projects, but in themselves, academic projects are not enough to achieve the goal and as Hazzan pointed out in [4], except in the studio method of teaching, there are very few interactions between students and teachers when students develop software systems. In order to bootstrap the repertoire for a given activity in SE (e.g. requirements analysis or design), we developed an approach based on the tailoring of activity before performing the activity itself. This approach will be presented through the specific case of the design in the rest of this paper.

2. Bootstrapping the repertoire

Let us illustrate the problem of gaining experience on the design process. "*The purpose of the Develop software design process is to establish a software design that effectively accommodates the software requirements; at the top-level this identifies the major software components and refines these into lower level software units which can be coded, compiled, and tested* [[5], part 2]". A baseline such as the ISO/IEC 15504-5 [6] (or ISO/IEC 12207 [7]) offers a set of outcomes of the process together with a set of base practices (or tasks) intended to accomplish outcomes.

For the Design Process, 12207 and 15504 outcomes are roughly the same:

1. a software architectural design is developed and baselined that describes the software elements that will implement the software requirements;
2. internal and external interfaces of each software elements are defined;
3. a detailed design is developed that describes software units that can be built and tested;
4. consistency and traceability are established between software requirements and software design.

SIGPLAN'05 June 12–15, 2005, Location, State, Country.

These outcomes and base practices are described at an abstract level identifying "what" should be done without specifying "how". The main difficulty for a project is to be provided with a response to "how ?" that is suited to the project specificities: technical, methodological, etc.

Although the repertoire structure can be drawn from the reference framework, when the activity is unknown, this structure does not take root and there is no chance that experiences (past or to come) could take place within the structure. The studio is the central training method in architecture schools and this analogy was used to provide a suitable educational environment for software design and development [8], [9]. Students work in teams on a large-scale project, supervised by faculty members, and generally for an external client. Behind software studios lies Donald Schön's idea of the reflective practitioner perspective [5], [10]. His proposition to educate the reflective practitioner is that students perform projects with a strong guidance of coaches. "*The experience of the students in the architectural studio, […] and, I believe, 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. The teachers can say things to them but they cannot understand what's meant at that point. The way at which they come to be able to understand what's meant is by plunging into the doing […] so as to have the kinds of experience from which they may then be able to make some sense of what it is that's being said.*" This education can be facilitated with our proposed approach intended to bootstrap the repertoire. The approach is generally accomplished in two steps: tailoring the activity - to acquire a minimal structure through a deductive approach (by writing a guide, for instance) then to initialize the repertoire through an inductive approach, for instance with the use of retro-engineering; and performing the activity - to begin to fill the repertoire whilst establishing the design of a real-scale project.

2.1 Tailoring the design process

However pertinent the reference framework may be, it must always be tailored to a project baseline. The methods and documents provided have a broad range. Tailoring aims to adapt the activities to be performed and products to be developed and delivered to the project's domain/size/methods/tools/etc.

When development environments are stable and mature, the way to use it can be found "off-the-shelf" or at least progressively built from successive projects, and capitalized in the corporate baseline. When methods and technologies are continuously evolving, the "software component" definition differs according to frameworks, models and tools. It is then necessary to constantly adapt the "how to design" issue to innovations and changes. This is precisely what the tailoring activities should provide answers to.

Tailoring is accomplished in two steps: study of the field (in order to establish the structure of the repertoire) then performance of a pseudo-design activity (in order to initialize the repertoire).

2.1.1 Establishing the structure of the repertoire: exploration of the design activity

The study of an activity can be accomplished through a preliminary work of thought and suggestion as to how to proceed in order to perform the concerned activity. For example, using a new method or tool begins with an exploration aimed at tailoring its usage to the specificities of the project. Each exploration ends with the writing of a usage guide or implementation guide for the activity in question. This kind of exploration activity favors and encourages student initiative and creativity on technical, methodological aspects or any activity of the software development process.

For example, exploring the design is aimed at understanding, preparing and defining the design model that should be used during the design phase. Thus, the instructions given in the "Design exploration" activity are intended to answer the following questions:

- what role does design play in our software process ?
- which determining elements and relevant models are useful in elaborating the design models ? why are they used ? how are they elaborated ?
- how do project design constraints (modelling language e.g. UML, process e.g. Unified Process, technical framework e.g. J2EE, ...) correlate to corporate baseline requirements ?

Once the design exploration has been performed, students asked themselves essential questions. Even though these answers may be wrong or unknown, this gave meaning to the structure of the reference framework. The framework can then be used as the first shape of the repertoire for this activity. For example, the ISO/IEC 12207 recommends the following tasks in order to perform the software architectural design:

1. transformation of the requirements for the software item into an architecture that describes its top-level structure and identifies the software components.
2. development and documentation of a top-level design for the interfaces external to the software item and between the software components of the software item.
3. development and documentation of a top-level design for the database.
4. development and documentation of preliminary versions of user documentation.
5. definition and documentation of preliminary test requirements and the schedule for Software Integration.

It is only at the point of performing design exploration that this tasks list makes sense to the student and can be used as a preliminary structure for his/her repertoire.

2.1.2 Initializing the repertoire: coding first then (retro-) design

Even once the "Design exploration" activity has been performed, design remains a difficult task. Plenty of books with case studies describe the design activity but leave the reader without a clear path to follow. Hence, our approach is to do NO design in the first instance - directly developing a prototype from requirements - , and to perform a retro-design of this prototype in later on. In the educational field, retro-engineering is an inductive approach. It is the reconstruction from back to front of a process, with the result of an activity as its starting point.

The "Retro-Design" activity thus helps answer the following questions:

- which model of our prototype can be helpful in understanding the code ?
- what elements can be established at the design phase ? how are they useful in implementing the design ?
- which tools can be helpful in expressing and managing the design, and generating programming artefacts ?

It is usually in doing the retro-design that students understand that the prototype implementation is badly structured, hard to develop … and it provides ideas for, and motivation to correctly re-design and re-implement the prototype. When the student's repertoire for a given activity is empty, any new task seems impossible. Retro-engineering provides a repertoire initialization.

2.2 Performing the design process

2.2.1 A first fill

Finally, students are able to perform the design of their project (capstone or real-life). Faced anew with the intended activity (the design), even it is perceived as new and unique, the student will see that something is still present and familiar in his/her repertoire. "*When a practitioner makes sense of a situation he perceives to be unique, he sees it as something already present in his repertoire. To see this site as that one is not to subsume the first under a familiar category or rule. It is […] to see the unfamiliar, unique situation as both similar to and different from the familiar one, without at first being able to say similar or different with respect to what. The familiar situation functions as a precedent, or a metaphor, or … an exemplar for the unfamiliar one*" ([3], p. 138).

The design is often presented as a shaping activity in order to give a form and architecture to the system that meet requirements. The learning of design objectives aims to elaborate a satisfactory design solution to a major problem whilst dealing with heterogeneous issues: technical, social, ergonomics … Learning by performing the design in a reflective education system relies on project-based work on complex and open-ended problems; includes frequent feedback and critique from tutors and peers; students are meant to learn to work as efficient members of a team. All these conditions are necessary in order to provide a real experience of design, which will take place in the repertoire previously set-up during the tailoring activity.

2.2.2 Life-long filling

"*Continuous learning reflects the notion that the pace of change in this modern age is such that an individual has to continually learn new things to keep up with the times, with a profession, or to be competent in any given job* [12]". Hence, the need to set up and develop a model of reflective acting and thinking during our initial education.

Reflection-on-action is an activity where we explore why we acted as we did, what was happening and so on. The notion of repertoire is very important in this approach. Practitioners build up a collection of ideas, examples, situations and actions and D. Schön saw this as central to reflective thought. There is a need for explicit and systematic management of knowledge and skills - and this has to be accomplished by professionals throughout their working lives.

We believe that lifelong learning should be – (i) - initially established through an appropriate education intended to develop a reflective attitude and – (ii) - then continuously sustained by periodic reflection-on-action activities.

3. Conclusion

D. Schön brought reflection into the centre of an understanding of what professionals do. Schön often associated the notion of repertoire with this theory. We stated in this paper that problem solving relies on the availability of a repertoire, which means: firstly giving a structure to the repertoire and initializing it, secondly beginning to fill it, and thirdly continuously sustaining the update of the repertoire.

Acknowledgments

The authors wish to thanks all students that learned design with this approach and whom feedback helped us to improve this learning path.

References

[1] Abran, A. and Moore, J. W. 2004. *Guide to the Software Engineering Body of Knowledge, 2004 Version* IEEE Computer Society http://www2.computer.org/portal/web/swebok/htmlformat (last accessed September 4th, 2009).

[2] ACM and IEEE 2004. *Software Engineering 2004* http://sites.computer.org/ccse/SE2004Volume.pdf (last accessed September 4th, 2009).

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

[4] Hazzan, O. 2002. *The reflective practitioner in software engineering education*. The Journal of Systems and Software 63, 161-171.

[5] Software Process Improvement and Capability determination 1995. *Software Process Assessment - Version 1.00*, http://www.sqi.gu.edu.au/spice/docs/baseline (last accessed February 13th, 2008).

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

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

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

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

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

[11] D. Schön, "Educating the Reflective Practitioner" in Meeting of the American Educational Research Association, 1987.

[12] Government of Canada 2009. Training Employees - Learning Concepts, http://www.hrmanagement.gc.ca/gol/hrmanagement/site.nsf/en/hr11570.html (last accessed September 4th, 2009).