Learning Software Engineering with EASE

Dirk Draheim
Institute of Computer Science, Free University Berlin, Takustrasse 9, 14195 Berlin, Germany

draheim@inf.fu-berlin.de

Key words: Didactics of Informatics, Higher Education, Teamwork

Abstract: The paper describes EASE - an ultra-lightweight software engineering process model that specifically targets student projects in higher education. The model has been obtained by analysing the requirements of student projects and then carefully combining relevant concepts from state-of-the-art software management concepts. Recommendations are given for structuring the course as a whole, as well as managing the work. The latter is oriented towards mature andragogical methodologies. EASE has significantly improved the learning outcomes in both undergraduate and graduate courses and the paper proposes that it should be adapted and trialled in secondary education and industrial training environments.

1. INTRODUCTION

EASE (Education for Actual Software Engineering) is a methodology for the teaching of software engineering skills in higher education. It applies to projects, i.e. every kind of course in which students build software systems by doing teamwork. Such projects belong to every core curriculum in computer science, both undergraduate and graduate. EASE consists of the following components (Figure1):

- driving forces;
- a process architecture;
- a practices catalogue.

In a broad sense the *driving forces* define the problem the lecturer is

The original version of this chapter was revised: The copyright line was incorrect. This has been corrected. The Erratum to this chapter is available at DOI: 10.1007/978-0-387-35663-1_34

T. J. van Weert et al. (eds.), *Informatics and the Digital Society*

faced with. The *process architecture* and the *practices catalogue* together constitute the solution. The practices form a *process pattern language* (Coplien 1995) (Figure 2).

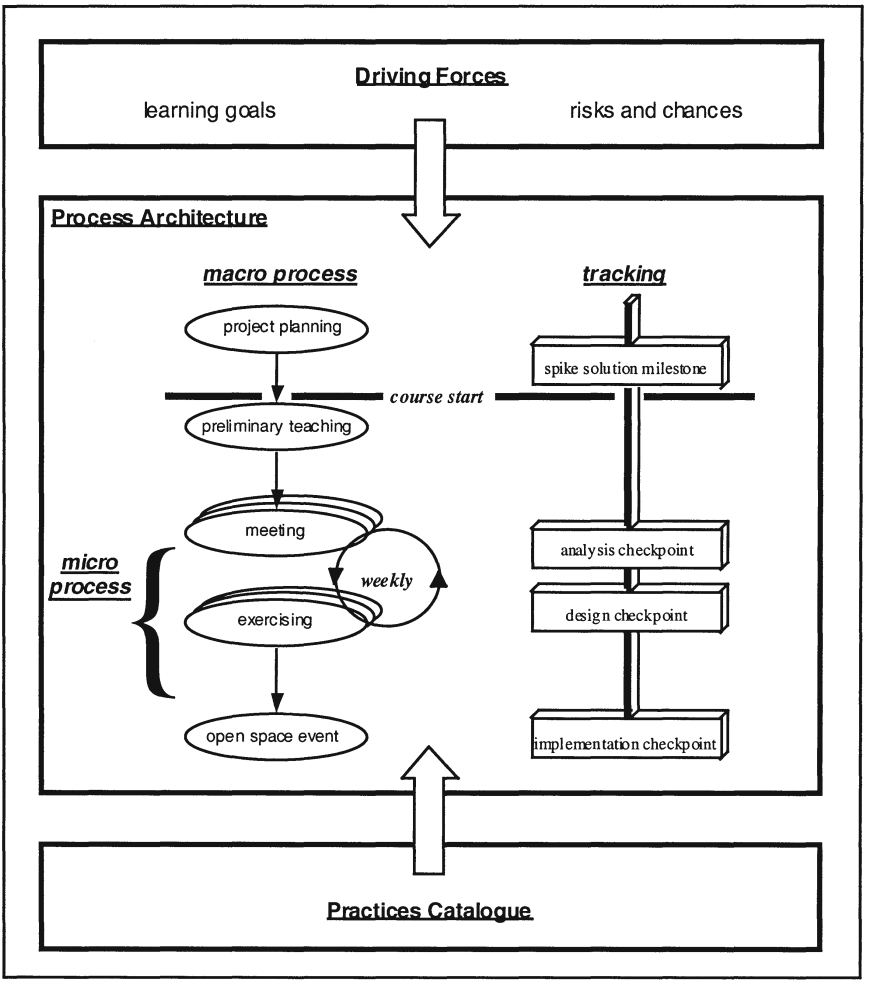

Figure 1. EASE - Education for Actual Software Engineering

EASE is the first process model that has been designed from scratch to accommodate the specific needs of education. Wherever appropriate EASE is oriented towards concepts found in today's software engineering process. With respect to didactics, EASE is influenced by collaborative learning (Brufee 1983).

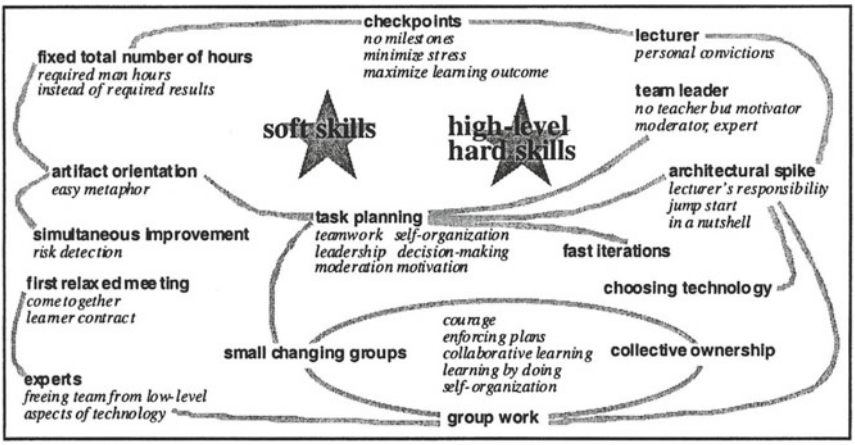

Figure 2. Practices pattern languages

2. DRIVING FORCES

We consider learning goals and risks and chances (or opportunities) are the driving forces of a software practical course. It is noteworthy that these driving forces are fundamentally different from those found in practice. Software management is a product-oriented task. The main driving force is risk and the entities to deal with during risk estimation are different from those in education - namely quality, quantity, cost, time and productivity.

A software course inherits risks and chances from adult education. There are also specific risks with regard to software engineering. Different individuals perceive the same topic in totally different ways (Varela 1988; 1993). Participants have different dispositions and hold different opinions about the purpose of computer science. As a result, the students have different objectives concerning their participation in the course. Like Siebert (2000) we regard this point as crucial for adult education and believe that the course can hardly be successful if it is ignored. However, we believe that is not only a risk factor but also a chance as different skills and opinions may contribute to collaborative learning (Brufee 1983).

These are chances. Students are receptive, hard-working and well-motivated. There is the risk that motivation may vanish. We believe it is very important to respect the students' different points of view in order to keep motivation high.

In nearly every software project, we observed students who only wanted to gather know-how about technology and were not interested in other learning goals. They have an attitude similar to that of real programmers and

sometimes "baby duck syndrome" occurs. Even more problematically, these students often tend to be non-communicative. They are not willing to work together with students that do not exactly share their opinions. In the worst case a project team consists of several small competing groups of programmers (and a bunch of increasingly intimidated other participants). With EASE's micro process, these students quickly become major contributors to teamwork.

Lecturers may introduce other risks when they simplify complex software engineering concepts. Sometimes concepts are simplified to such an extent that students are misled. Take object orientation as a prominent example - often is taught as a modelling the world with objects paradigm, without caveats concerning the new impedance mismatch between problem domains and solution domains of non-trivial systems, that is the result of object orientation.

EASE is designed to mitigate risks and exploit chances. Therefore EASE fosters teamwork and collaborative learning, self-organization and learning by doing.

Currently EASE uses its own, quite coarse, taxonomy of learning aims. These include *hard skills* which are:

- technology-related, low-level: detailed know-how about concrete technologies, e.g. programming languages, APIs, operating systems, CASE tools;
- technology-related, high-level: the ability to recognize advantages and disadvantages of concrete technologies with respect to a current ambiguous task;
- method-related, low-level: know-how about concrete software engineering techniques and concepts, e.g. specification languages, modelling languages, process models, design patterns, GRASP patterns;
- method-related, high-level: the ability to appreciate the respective importance of concrete software engineering techniques and concepts.

There are also *soft skills*:

- courage, skills concerning teamwork, leadership, decision-making, enforcing plans, motivation, and moderation.

Finally there are *key skills*:

- receptiveness, efficiency, ability to learn.

EASE is supposed to support the learning of soft skills and high-level hard skills, but it does not prescribe concrete learning goals. For instance, in a particular project modelling, specification, design, or even coding may be emphasized. EASE provides a framework for learning and exercising (or applying) software engineering. The lecturer defines the learning goals.

3. PROCESS ARCHITECTURE

The core process of EASE has a sophisticated, yet easy to learn architecture. It consists of a macro process that is accompanied by tracking (Figure 1). The macro process starts with a project planning phase and is followed by a preliminary teaching phase. Then a controlled iterative micro process is entered, which consists of repeated steps, namely meetings and exercises. The micro process takes the majority of the project's time. The project is closed with an open space event. Tracking encompasses a spike solution milestone and checkpoints for analysis, design, and implementation.

EASE's meeting concept was developed by adapting the iteration planning of Extreme Programming (Beck 1999), and it is inspired by Collaborative Learning (Bruffee 1983) and Action Learning (Revans 1982). The resultant micro-process iteration bore a striking similarity to the session workflow of Entraînement Mental (Chosson 1975). We regard this as a further justification of our approach. EASE, in common with the USDP (United Software Development Process) (Jacobsen et al 1999) has a controlled simultaneous improvement of artifact sets, termed two-dimensional process structure in USDP.

3.1 Macro Process

Project Planning

Some concepts in EASE presuppose that the course has enough participants to be divided into several teams, so that each team has ten to forty members. Each team has a lecturer or a tutor as their leader. With minor changes EASE can be used in courses of smaller scope.

The lecturer prepares the course defining learning goals based on personal convictions. Learning goals may be predefined because of an encompassing study plan. The lecturer defines the desired system as a vision document. Each team is set the task of realizing as many features as possible. Work is finished when every student has worked a fixed total number of man hours - a figure agreed at the beginning of the course. The course is finished with a checkpoint - not by a milestone as that enforces a predefined feature set. The lecturer defines sets of simple artifacts - EASE task planning is artifact-oriented. These are divided into an analysis set, a design set and an implementation set - an intentionally coarse division. In contrast to the theory of stage-wise or waterfall-like software management, EASE assumes that artifact sets are developed simultaneously from the outset - simultaneous improvement. EASE is deliberately not prescriptive at the level of engineering activities in order to avoid micro management. Instead it is designed to be an instructive challenge for the students. They must find the

tasks that contribute best to progress on their own.

The definition of the artifact sets is accompanied by choosing technologies and techniques and by developing a system architecture. For this purpose the lecturer has to develop an architectural spike. This is a prototype system which realises just one or a few representative use cases of the desired system. All the technologies on which a chosen system architecture is based are used. There are subtle differences between the reasons for architectural spikes in EASE and those found in practice. The purpose in practice is for estimating risks associated with a particular architecture. In EASE the resulting spike solution is used to present the chosen architecture and technologies to the students in preliminary teaching. It serves as both start point and reference point during the rest of the course. The spike solution must consequently avoid encompassing unnecessary features like exhaustive functionality, constraint checking, and fancy user-interface layout. Using a spike this way has the following benefits:

- "in a nutshell" - the students understand the architecture as a whole.
- "jump start" - the students have early success in handling technology.

In this version of EASE, the students do not develop the architecture. This is considered too challenging. In practice it is a senior architect's task. The lecturer fixes days for the analysis and the design checkpoint.

Preliminary Teaching

In a first plenum the desired system is explained. EASE is explained, especially the micro process. An overview of the architectural spike is given. The students are assigned to teams randomly. Homogenous groups should be avoided. An exception may be made if there are requests for homogenous female teams. The students of each team get together in a first relaxed meeting and a first simple task is assigned to the students.

Micro Process

Each team enters the micro process, which is fast iterative and consists of alternating weekly meetings and exercises. Meetings are for task planning. During exercise periods the tasks defined in the meeting are carried out. At each meeting the whole team and its team leader get together. This team leader is the motivator, as required a moderator, occasionally an expert.

A meeting takes approximately one and a half hours. During the meeting five well defined activities take place (Figure 3). It is important to note that in the first few meetings these activities are viewed as sequential steps that define a workflow. Once students are used to the meetings activities should be carried out in parallel. In this way, the mutual dependencies between the activities are exploited to improve the outcome. However, the workflow technique may be used if teamwork gets stuck. For better understanding activities are presented as workflow steps in this paper and that is the way they should be explained in preliminary teaching.

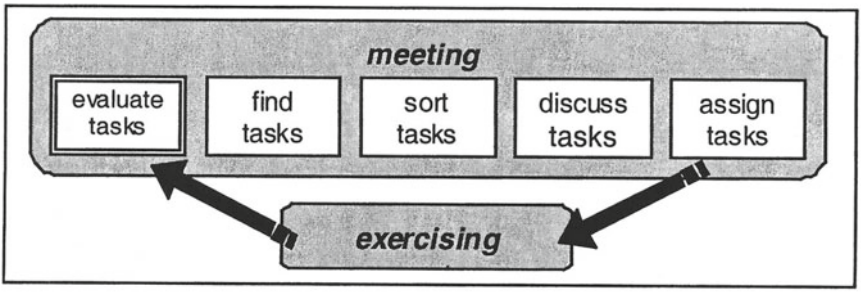

Figure 3. EASE - micro process

First, the results of the previous week's effort are discussed and validated. Then the team concentrates on finding new tasks. Task planning is artifact oriented. The main question is - what has to be done next in order to contribute most effectively to the growth and the quality of the artifact set? Brainstorming techniques are very helpful. All suggestions must be taken seriously and gathered for further discussion. Usually more work is found than is possible to do during the following week so tasks have to be sorted with respect to importance. Based on their current knowledge and opinions the tutees discuss what to do next and why. This results in a task list, ordered by importance. Students learn to appraise the relative importance of the mutually dependent families of software engineering activities.

Alternative solutions for the tasks are suggested, discussed and finally one of them is chosen for each task. Tasks that are too large to be carried out by a group in a week have to be split into several tasks. In the end every task is assigned a new, succinct title and a short, nonetheless precise, description. Tasks then have to be assigned to team members. Usually a task is assigned to a group of two to four participants. Group members estimate the time needed to carry out the task and agree the time contribution of each member. Tasks that cannot be assigned are considered the following week.

The team is divided into small groups that work on tasks. This division is not static and at each meeting, new groups are formed. The tutor has to monitor the selection. In a project threads of strongly dependent tasks exist. If a task has a predecessor, it should be assigned to the predecessor's group but at least one group member should be replaced with a new team member. This concept of small and frequently changing groups is crucial, because we pursue a generalized notion of collective ownership. It is the target of the exercise that students learn about the overall structure of the system that is built up. Ideally, every student should have an understanding of every technology and concept used in the project, and how they interact. Consequently students do not over-specialize and the learning outcome is

shifted in the direction of high-level hard skills and soft skills.

The meeting is followed by exercises. Tasks are carried out using the teamwork approach. Often the group work can be further sub-divided into smaller tasks for every group member - nearly always true for coding tasks. Even with this sub-division the group should stay together in the same room, although every team member may be working on their own. If a problem arises all group members immediately work together to find a solution. Group members help each other on demand. If the task is coding, pair programming is recommended. If a new group member joins every question has to be answered patiently. The concept of small changing groups makes tutees learn from each other. At the next meeting results are evaluated. Students talk about their experiences, describe problems that occurred and explain how long they worked (they have to work a fixed total number of hours). Independent of the total outcome of work we find students typically are willing to respect the work of other, perhaps less experienced, students. Our approach tackles the problem that students often do not trust the industry of their colleagues because they have different capabilities - a potential source of squabble. The overall teamwork should be supported by appropriate simple tools like repositories, mailing lists and discussion forums. The course terminates with *an open space event*. This realises the implementation checkpoint. Each team explains its system to the other teams in a detailed plenum presentation. Then all students gather in computer rooms to examine and test all the systems.

3.2 Tracking

On the fixed days of the analysis and design checkpoint, the respective artifact set is reviewed by the team leader. In a plenum, results are presented briefly, compared and discussed. We use the term checkpoint instead of milestone. A milestone defines a targeted result. A checkpoint is a softer control mechanism and fixes a day for reflection and problem exploration.

3.3 Case Study

We developed EASE based on the experiences gathered in our lectures on software engineering and their satellite projects. EASE was thoroughly used for the first time in summer 2001 in the lecture "Softwarepraktikum" (undergraduate, 4 semester hours) at Freie Universität Berlin. It significantly improved the learning outcome. It was used by three teams each consisting of fifteen to twenty tutees. Only eighty per cent of the participants studied computer science. Ten per cent had extensive skills in net programming or

using databases, usually only low-level hard skills. At the beginning of the course the other students had hardly any knowledge about distributed systems or persistent data. By the end of the course every student could deal with the technologies and several complex notions shown in (Figure 4).

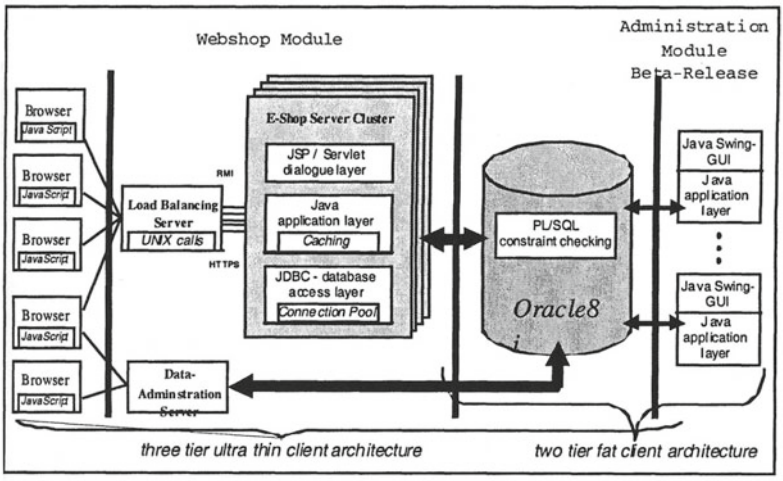

Figure 4. Case study - system architecture

The artifact sets encompassed a web user interface prototype as the result of storyboarding, a UML data model, a textual requirement specification, deployment and design diagrams, commented code and automated tests. The students learned using SourceForge's open source management capabilities, Concurrent Versioning System software, modelling tools, Integrated Development Environments and GUI builders.

4. FURTHER DIRECTIONS AND RELATED WORK

We recommend EASE should be adapted to other areas, e.g. secondary teaching or industrial training. In a graduate course it should be possible for students to take over the project-planning phase and are evaluating this in a challenging practical course on EJB (Enterprise Java Beans) application servers. A web-enabled, team-oriented EASE project management tool is under development. Lessons learned from using XP and the Rational Unified Process (RUP) in student projects are presented in a number of papers (Lippert et al 2001; Traub 2001). In our opinion it is not possible to use XP in a student project. Teaching XP using just a few XP notions misses the point. It is a common misunderstanding that XP is an ultra-lightweight

process model. Instead it defines a highly sophisticated net of practices that can be exploited only in a real world project. Unsophisticated attempts to use XP in education can seriously mislead students (nonetheless XP can be taught - by using usual "frontal" teaching). This applies even more so for the heavyweight RUP (Kruchten 1999). The Unified Process for education UP/EDU (Robillard 2001) is the result of reducing RUP so it can be used in education. The result is still overpowering as UP/EDU compels the lecturer to teach certain topics, particularly subtle proprietary RUP notions.

If a professional process model is used in education naïvely or in an oversimplified way, its constituting concepts are probably not exploited at all. A student project differs from a real world project in scope, in driving forces and in organisational culture. EASE is a reusable software engineering process for practical courses in higher education. It is ultra-lightweight and combines new concepts with proven concepts from software management and didactics. EASE targets the learning of soft-skills and high-level hard skills.

REFERENCES

Beck, K. (1999) *Extreme Programming Explained - Embrace Change*. Addison Wesley.

Bruffee, A. (1983) *Collaborative Learning: Higher Education, Interdependence, and the Authority of Knowledge*. Johns Hopkins University Press.

Chosson, J.-F. (1975) *L'entraînement mental*. Le Seuil.

Coplien, J. O. (1995) A Development-Process Generative Pattern Language. In *Proceedings of1st Pattern Languages of Program Design*, J. O Coplien and D. C. Schmidt (eds.), Addison-Wesley, pp. 183-237.

Jacobson, I., Booch, G. and Rumbaugh, J. (1999) *The Unified Software Development Process*. Addison-Wesley.

Kruchten, P. (1999) *The Rational Unified Process*. Addison-Wesley.

Lippert, M. et al (2001) XP lehren und lernen. In *Software Engineering im Unterricht der Hochschulen*, H. Lichter and M. Glinz (eds.), dpunkt.verlag.

Revans, R. W. (1982) What is Action Learning ? The *Journal of Management Development*, Vol. 1, No. 3, pp. 64-75, MCB Publications.

Robillard, P. N., Kruchten, P. and DíAstous, P. (2001) YOOPEEDOO (UPEDU): A Process for Teaching Software Process. In *Proceedings of the 14th Conference on Software Engineering Education and Training*. IEEE.

Siebert,H. (2000) *Didaktisches Handeln in der Erwachsenenbildung - Didaktik aus konstruktivistischer Sicht*. Luchterhand.

Traub, S. (2001) Einsatz von objektorientierten Technologien und Softwareentwicklungs-prozessen in der Lehre. In *Proceedings of NetObjectDays 2001*. TranSIT.

Varela, F (1988) *Cognitive Science - a Cartography of Current Ideas*. Pergamon Press/Leuven University Press.

Varela, F. (1993) *Kognitionswissenschaft - Kognitionstechnik*. Suhrkamp.