Annales UMCS Informatica AI 4 (2006) 34-44

Annales UMCS
Informatica
Lublin-Polonia
Sectio AI

http://www.annales.umcs.lublin.pl/

Experiences of software engineering training

Ilona Bluemke*, Anna Derezińska

Institute of Computer Science, Warsaw University of Technology,
Nowowiejska 15/17, 00-665 Warszawa, Poland

Abstract

In this paper some experiences from laboratories of Advanced Software Engineering course are presented. This laboratory consists of seven exercises. The subjects of exercises are following: requirements engineering, system design with UML [1], reuse, precise modelling with the Object Constraint Language (OCL [2]), code coverage testing, memory leaks detection and improving application efficiency. For each laboratory exercise a set of training materials and instructions was developed. These internet materials are stored on a department server and are available for all students and lecturers of this course. Rational Suite tools [3] are used in the laboratory. The goal of introducing Internet materials was to improve the quality of SE education. Some experiences and observations are presented. The evaluation of students results is also given.

1. Introduction

Much has been already done in the area of teaching software engineering. Every year there are held international conferences on this subject, e.g. Conference on Software Engineering Education and Training, or during the conferences on software engineering there are special sessions, e.g. Automated Software Engineering, Software Engineering. In the proceedings from these conferences many interesting ideas or curricula can be found, e.g. [4-6]. In this paper we present some experiences from advanced software engineering laboratory introduced in the Institute of Computer Science in fall 2004. In the Department of Electronics and Information Technology, Warsaw University of Technology two software engineering courses are taught every semester. The first one, called Software Engineering, is given to the students in the fifth semester and the second one – Advanced Software Engineering (ASE), is given to the same students in the sixth semester. During the Software Engineering course students learn the basic ideas of software engineering and object modelling in UML (the Unified Modeling Language [1]). They also spend 15 hours in laboratory preparing a project of a simple system in Rational Rose.

*Corresponding author: *e-mail address*: I.Bluemke@ii.pw.edu.pl

The Advanced Software Engineering course should broaden students' knowledge of software engineering. To improve the quality of education in the ASE course the laboratory was enhanced with Internet support. Choosing subjects for ASE course, the directions from Software Engineering Body of Knowledge SWEBOK [7] were considered. It was not possible to use all the directions from SWEBOK and present all aspects of software engineering in 30 hours of lectures. The chosen subjects are the following:

- requirements engineering,
- design with reuse,
- precise modelling,
- testing,
- quality improvement,
- software effectiveness,
- software processes.

The subjects shown above relate to six out of ten Knowledge Areas (KAs) proposed by SWEBOK [7].

In the ASE course students have 15 laboratory hours in blocks of two. Seven laboratory subjects, precisely connected with lectures, were prepared. In the Institute of Computer Science every semester about sixty students, divided into groups of 7-8, have ASE laboratory. Five instructors supervise these groups. When there are several supervisors students often complain that their demands, requirements and grades are not the same. In order to overcome this problem detailed instructions were written for each laboratory session. While using these instructions the laboratory sessions supervised by different lecturers are similar and the time spent in laboratory can be used very effectively.

The goal of ASE laboratory was also to present the professional CASE tools supporting different phases of the software process. The tools from Rational Suite [3] used during laboratories are Requisite Pro, PureCoverage, Quantify and Purify. These tools are complicated and it is difficult to use them without training. Rational tutorials for these tools, available in English, are rather long and can not be used during two hours of laboratory. A set of tutorials for these tools was prepared in Polish, native language of the students. The tutorials and the laboratory instructions are stored on the department server and are available to students, lecturers and instructors of the ASE course. Each student can study the instruction and tutoring material at home, so the instructor's directions can not surprise her/him.

In Section 2 Internet materials are briefly presented. The subjects of ASE laboratory are presented in Section 3 and illustrated with examples. Evaluation of laboratory results is discussed in Section 4.

2. Internet materials

The ASE laboratory materials are written in html and stored on the department server. Only students and lecturers of this course have access to these materials. The structure of ASE laboratory materials is following:

- Introduction containing ASE laboratory goals
- Tutoring materials:
  - Requisite Pro
  - Code Coverage
  - Purify
  - Quantify
  - OCL
- Design Patterns
- Format of use – case description
- Laboratory directions
- Vocabulary
- Bibliography.

The organization of these materials is hierarchical. On every screen there are links to the "parent" section and the main index, to the next and previous sections. In each laboratory instruction there are links to the appropriate tutorial material and positions in vocabulary (Fig. 1).

3. ASE laboratory

The main goal of ASE laboratory was to refine students' knowledge and experience. Seven laboratory exercises were developed. The subjects of these exercises are as follows:

1. Requirements engineering
2. Software design
3. Reuse
4. Precise modelling with OCL
5. Code coverage testing
6. Quality improvement
7. Improving effectiveness.

Subjects of exercises one and three to seven are completely new to students. In all laboratory exercises the tools from Rational Suite: Requisite Pro, Rose, PureCoverage, SoDa, Purify, Quantify are used. Students in the sixth semester are familiar only with Rose and SoDa, because these tools were used in the software engineering laboratory in the former semester. Rational Suite provides tutorials (in English) for most tools. Each of these tutorials needs at most two hours, so they are too long to be used during two hours in laboratory. During the ASE lectures there is no time to present and discuss CASE tools in detail.

Problems with a tool usage could disturb the student in the development of a laboratory subject.

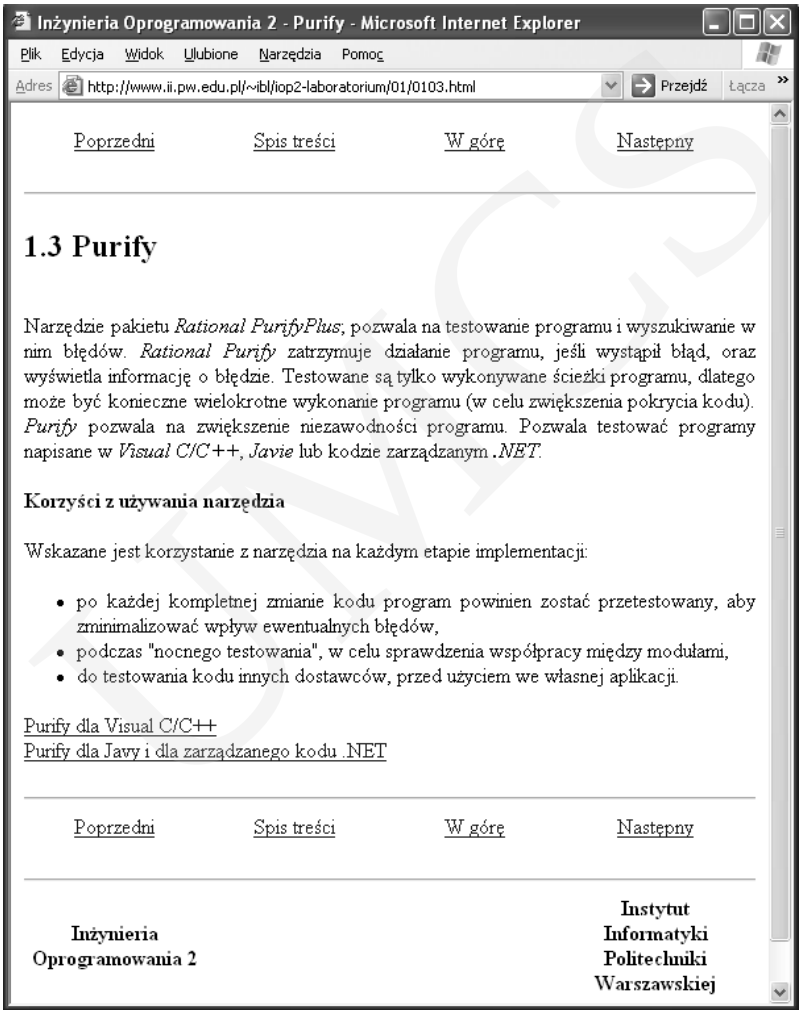

Fig. 1. Screen from ASE the laboratory system

To overcome difficulties caused by using several CASE tools, short tutorials in Polish, native language of the students, were prepared. All these tutorials are stored on the department server and are available to the ASE students the whole semester. Although OCL and the design with reuse are presented during the ASE lectures, some tutoring materials for these subjects are also available.

Students often complain that laboratories conducted by different instructors are not the same. To solve this problem detailed instructions for each laboratory exercise were written and stored on the department server as well.

Each of laboratory instructions consists of three parts:

1. Introduction
2. Detailed directions
3. Exercise results.

In each laboratory instruction there are links to tutorial materials, bibliography and description of difficult ideas in the vocabulary part. Introduction describes the goals of the exercise and the basic concepts. The tasks to be prepared before a laboratory exercise can be defined in the introduction. The second part contains a sequence of tasks that should be executed during the laboratory session. Finally, the expected results and the evaluation criteria are given.

In Section 3.1 the translation of a part of the instruction "Requirements Engineering" is presented. Section 3.2 describes the organization of laboratory and in Section 3.3 discrepancies often met by the instructors are shown.

3.1. Example of laboratory instruction

Below, an extract from a translated instruction for the first ASE exercise, concerning requirements engineering, is presented. *Italic* font is used to denote tool buttons, options and underline shows links to other parts of ASE internet materials, dots (...) indicate the omitted parts of the instruction:

Introduction

The aim of the exercise is the development of requirements specification for a given system and training in requirements management [8]. The requirements management covers the following activities: systematic eliciting, organizing and documenting requirements. In order to preserve the contract between the client and the development team, the changes in requirements must be handled. Requirements specification should follow the quality criteria (IEEE 830 [9]). The requirement repository is created using RequisitePro, and a use case model is created using Rational Rose.

Tasks

1. Create a requirements repository using Requisite Pro (...)

Then, iteratively execute steps from 2 to 4.

2. Define different program features, stakeholder requests, non-functional and functional requirements. Create requirements in the appropriate packages, e.g. use case. (...)
3. Specify requirements
   - Define stakeholders for stakeholder requests.
   - Define requirement attributes: priority, difficulty, stability, cost.

- Create hierarchy of requirements, e.g. for requirement UC2 its descendants UC2.1 and UC2.2 (use option *New->Child Requirement*).
- Connect requirements with traceability links. Create traceability matrix. For a given package choose New->View and select Traceability Matrix in View Type. (...)

4. Create a use-case model
   - Create a new UML project in Rose.
   - Generate use cases and associate with the requirements of the type *Use Case* from RequisitePro (...). Adjust requirements' text with the use case names. (...)
   - Draw use-case diagram(s) in Rose. Create actors and connect them with the appropriate use cases.

   Add requirements description for the use cases in *Documentation* fields in Rose. The formatted pattern of a use case description can be followed.
5. Generate documentation files (...).

Results

Final results of the laboratory comprise three documentation files, indicated above, and the use-case model. (...).

It should be noted that the instruction includes mainly information concerning the CASE tools and specifying the artifacts to be prepared by the students. Other activities not described in the instructions are also necessary. For example, during an exercise on requirements engineering students should negotiate the goals of his/her project with others in the same group (see Section 3.2). We assume that the instructor is responsible for stimulating these activities during laboratory exercises. It seems to us, that too long and detailed instructions are not effective.

3.2. Laboratory organization

The ability to work in a team is very important for a software engineer and should be trained as well. Interesting proposal how to practice the teamwork can be found for example in [10]. However, developing the ASE laboratory we were aware of the significance of the teamwork, we decided to incorporate such training in this laboratory only in a very limited extent due to the time constraints (only 15 hours of laboratory). In ASE laboratory one instructor – lecturer has a group of seven to eight students. A group of students prepares requirements for a common subject, divided into individual subtasks. The members of the group collaborate with each other defining the relations between the subtasks. To show some teamwork problems, the specification of requirements prepared during the first laboratory is reviewed by another student.

The results of the reviewing process are shown in Section 3.3. During the second and third laboratory classes, the reviewer designs the system using the specification reviewed by her/him. Some OCL specifications should be inserted in the project, for example in the specification of an operation. The use of design patterns is also recommended [11].

While doing the exercises one to three, the instructor comes up to each student and points out the weak points in the design or specification. It usually takes all the time available for him/her in the laboratory and it requires a lot of effort.

Exercises five, six, seven are devoted to the code coverage testing as well as to quality and effectiveness improvement. In these exercises students, grace to detailed instructions, can work more independently. Therefore these exercises can be done in a "remote" manner.

A student has to send his/her exercise results to the tutor by email. The instructor evaluates the received artifacts (remarks, specification, projects, code, etc.) and inserts the results into the department server. At the beginning of the next laboratory session the instructor points to all errors made by the student. Though such evaluation is time consuming we hope it is worthwhile. The obtained results of ASE tests show much better scores than in the previous semester; without Internet support and with on-line exercise evaluation.

3.3. Discrepancies in requirements engineering

The analysis of artifacts obtained from students is time consuming. Many factors are relevant. Below, some factors taken into account while evaluating exercise "Requirement engineering" are listed:

- correctness, consistency and completeness of requirements set,
- logical categorization of features (goals), stakeholder requests and use-case requirements,
- specification of stakeholder requests:
  - clear and complete identification of stakeholders,
  - complete and proper assignment of stakeholder requests to stakeholders,
- identification of non-functional requirements and its specification,
- logical distribution of requirements to different levels of requirements hierarchy,
- definition of attributes of requirements:
  - assignment of requirement priorities,
  - assessment of realization difficulty,
  - recognition of stable and unstable requirements,
- clear and unambiguous requirements description,
- definition of traceability relation between requirements:
  - adequate selection of traceability relations,

  - concise directions of traceability arrows.
  - completeness of traceability relations,
- association of requirements to appropriate use- cases in the use-case model,
- quality of use-case diagrams:
  - proper identification of an actor,
  - correct syntax of relations between use-cases,
  - adequate usage of different relations,
  - clear structure of use-case model,
  - readability of the diagrams,
- quality of use-case description (a detailed pattern was given in training materials):
  - identification of the realization flows and consecutive steps,
  - adequate abstraction level of step description (functional description versus an interface description),
  - clear pre- and post-conditions (if necessary).

It should be noted that our students already prepared a simple use-case model in the previous semester. In the present semester the students properly recognized properly the general concepts of use-cases, but still made some elementary syntax errors in the use-case diagrams.

The final verification of the requirements is a working system and satisfaction of its users. During the teaching process verification should be done immediately and a student should obtain a feedback about his/her work. Therefore, the results of the requirements engineering task are read by an instructor and also reviewed by another student. The following artifacts are evaluated: documentation with a description of requirements, a hierarchy and traceability relations, and a use-case model. We were surprised that students were able to find many errors in their reviews. The errors were often similar to those they made in their own requirements specification. Below a list of errors recognized by students while reviewing the requirements prepared by a colleague is given. The numbers given in brackets show in percentage in how many reviews errors were detected.

- Missing requirements (64%).
- Unclear or missing traceability relations (45.5%).
- Different qualification of features and use-case requirements (45.5%).
- Lack of functionality, which should be available for a recognized stakeholder/user (41%).
- Ambiguous definition of requirements (41%).
- Assignment of requirements to different modules of the system (36%).
- Incorrectly assigned priority to requirements (36%).
- Improper usage of relations between use-cases (include, extend, generalization) (32%).
- Unclear definition of stakeholder requests (32%).

- Insufficient description of a use-case, missing desired pre- and post-conditions (32%).
- Illegible use-case diagrams (23%).
- Inadequate assessment of the realization difficulty (23%).
- Missing actors in use-case model (23%).
- Lack of defined requirements attributes - importance, stability (18%).
- Incorrect hierarchy of features or use cases (9%).
- Ambiguous name of an actor (9%).
- Superfluous requirements (4.5%).

In general, students were able to identify correctly weak points in the requirements descriptions, even, if they made the same errors. From the list given above it can be seen that students detect many discrepancies from the list shown at the beginning of this section.

4. Laboratory evaluation

Objective evaluation of students' skills is very difficult and requires a lot of time. In the ASE laboratory a tutor has a relatively small number of students (seven to eight). Evaluation of all laboratory exercises is based on:

- An activity of a student, including the approach to the solutions and presented partial results during in laboratory classes.
- Realized tasks (specifications, models, programs) and final remarks received via email.
- A critical discussion about the obtained results.
- A review by a colleague (only for the first exercise).

The obtained results contributed to the score of an exercise. Figure 2 shows the percentage of points given to students by four instructors in spring 2005.

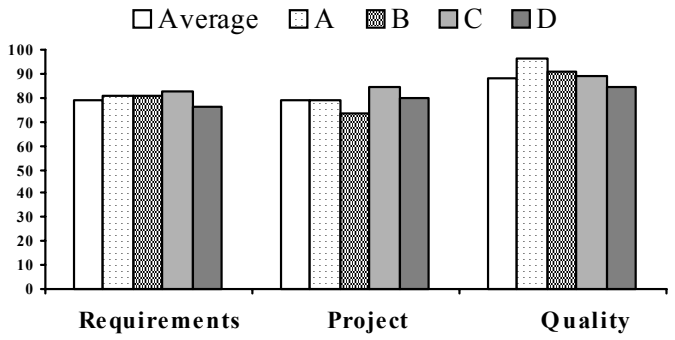

Fig. 2. Laboratory results

Three parts of the laboratory were evaluated independently. On the average the results of the first part – the requirements engineering, and of the second

part – the project with elements of reuse and OCL, were about 79%. The results of the last part concerning testing, quality and performance analysis (exercises five, six, seven – section 3) were higher (88%). The subjects of these exercises required many manipulations with the new tools (this could be easily done using the instructions and tutorials) but a less creative work than subjects of exercises one to four. Moreover, the students were more capable and willing to cope with any problems related directly to the running programs than to those concerning requirements and models. The analyzed programs were not very complicated, because the main goal was to learn the new methodology.

The Advanced Software Engineering course is given to students of two specializations: Engineering of Information Systems (EIS) and Information and Decision Systems (IDS). All students attend the same lectures and have to pass the same exams. EIS students have ASE laboratory supported by Internet materials presented in this paper and IDS students make a project. IDS students can also use Internet materials. The laboratory and the project cover similar areas of software engineering and give the same number of credit points.

Table 1. Exam results

Exam subjects		Results of students	
		after ASE laboratory	after project
Autumn 2004/2005	Requirements	40%	35%
	UML/OCL	49%	34%
Spring 2005	Requirements	93%	81%
	UML/OCL	63%	33%
Sum		62%	40%

In Table 1 the results of exams in two semesters are shown. The number of points obtained by students, divided by the maximal amount available, is given in percentage. From the exam only the results of tasks concerning subjects practised during the laboratory and the project were selected. Internet materials are available to all students. During lectures all students were encouraged to study these materials. In two separate columns the results of EIS students – having the laboratory are compared with the results of IDS students – having the project. It can be noticed that students having the laboratory, forced to study our Internet materials, achieved significantly higher scores.

5. Conclusions

The need to include a wide scope of subjects in very limited time motivated us to prepare an Internet support for the ASE course (about 90 pages). This benefited in significantly increased quality of SE education. As the results of the exam showed students attending the ASE laboratory achieved higher scores than

those having projects, who also have access to these materials and even were encouraged to use them.

The ASE laboratory comprises phases of software process like the requirements engineering, system design with reuse and design patterns, precise modelling with OCL, code coverage testing, and improving the quality and effectiveness of application. Usually, these subjects are taught in separate courses and do not appear together in curricula of a single, obligatory software engineering course.

The ASE laboratory was introduced in autumn 2004 and about a hundred students finished it. So far, it has been observed that the students were able to do much more than in the previous semester without Internet instructions. The demands of different instructors were similar. Also the average grades were close. Instructors were able to devote more attention to the design assessment and refinement; they did not waste time explaining detailed steps of each exercise.

Our intention was to use industrially proven tools in the ASE laboratory. By providing the tutoring materials a student could take an advantage of the information they provide without being exposed to the hassles of learning to use the tools. Students were satisfied working with professional CASE tools. They also get some practical experience considering software design, implementation, testing and effectiveness.

We have shown that in a short time, they can gain experience broadening their understanding of engineering a program.

References

[1] Unified Modelling Language Specification, www.omg.org/uml
[2] Warmer J., Kleppe A., *The Object Constraint Language Precise Modeling with UML*. Addison Wesley, (1999).
[3] Rational Suite, http://www-306.ibm.com/software/ rational/
[4] Jazayeri M., *The education of a Software Engineer*. Proceedings of the 19th Inter. Conf. on Automated Software Engineering, ASE'04, (2004).
[5] Rombach D., *Teaching how to engineer software*. Proceedings of 16th Conference on Software Engineering Education and Training, CSEE&T, (2003).
[6] Nikula U., *Experiences of embedding training in a basic requirements engineering method*. Proc of 17th Conf on Software Engineering Education and Training, CSEET'04, (2004) 104.
[7] www.swebok.org, Guide to the Software Engineering Body of Knowledge, (2004).
[8] Leffingwell D, Widrig D., *Managing software requirements*. A unified approach, Addison Wesley, (2000).
[9] IEEE Std 830-1998, IEEE Recommended Practice for Software Requirements Specifications.
[10] Nawrocki J.R., *Towards Educating Leaders of Software Teams: A New Software Engineering Programme at PUT*. in P.Klint, J.R.Nawrocki (eds.), Software Engineering Education Symposium SEES'98 Conference Proceedings, Scientific Publishers OWN, Poznań, (1998) 149.
[11] Gamma E., Helm R., Johnson R., Vlissides J., *Design patterns: Elements of reusable object-oriented software*. Addison-Wesley, (1995).