TEACHING RELIABILITY AND SCHEDULE CONTROL
(DRAFT)

KRZYSZTOF KACZMARSKI

ABSTRACT. The paper describes a way of teaching reliability and schedule control in software development by constant, well organized and well defined interaction between educator and student teams. Selected tasks leads to certain development problems, which are also trying to prepare students for real life.

1. INTRODUCTION

The PION Software Engineering Laboratory is a continuation of Software Engineering Lectures delivered at our Faculty on the fifth semester of BSc studies. Labs are then placed on the sixth semester filling it with 15 meetings of 3 hours (should be extended to 6 hours). The goal of the labs is to teach teamwork in some real industrial constrains like: time/coast limit, complicated and complex environment and reliability expectations.

The course and labs are organized by Krzysztof Kaczmarski and Andrzej Wąsowski with wardship of prof. Bohdan Macukow.

1.1. Goal of the courses.

1.1.1. Time/cost constrains. In most of the projects developed by students during studies there is no cost/time control. Participants often just choose a project, and then must develop it. They usually write applications at home or in university labs but they do not control how much time do they spent working on it. They independently organize their time, so we actually do not know if they make good time/organization decisions. After some time students simply bring finished (working or not) applications and present documentation.

Experience shows that they often spend much more time on a project than it is necessary. That happens because of several reasons (to enumerate only a few of them):

- bad organization of work
- bad development decisions
- technical problems
- lack of a good schedule
- not obeying a schedule
- inter-team conflicts

We have tried to force them to use a simplified Personal Software Process (PSP) in their homework, however with no success. At the beginning students agreed that they should control their time but then usually forgot to fill in a vary simple time/cost form or made it in a wrong way not documenting anything.

Our investigations (and self experiences) proved that students often spend a lot of time on not important parts of the project, noticing real difficulties in the most unwanted moment, which leads to unexpected problems and time delays. They also usually make some accidental movements which often do not help with certain problem.

One of the most important goals of the presented teaching pattern is to force students to control their time – use systematic time schedule forecasting, and methodology – choose and follow one. A good students’ system design becomes a basis for the whole time schedule.
1.1.2. **Complicated environment.** Real industrial projects usually join several technologies. Web based applications, real-time systems, databases, xml technology, etc. use many different languages, precompilers, tools, configuration files etc. Students, future software developers are usually not familiar with most of them. We believe that it is not possible to give them even an incomplete knowledge of the selected most important technologies. It is far beyond the possibilities of 5 year studies which must give background of many other disciplines. The only way to prepare students to work in this fast-changing branch of industry is to teach them how to learn fast basing on some (probably partial) documentation.

There is then another goal to teach students to incorporate unknown technologies as fast as possible, by recognizing the basic functionality. They should understand new concepts by connecting or comparing them to some already known ones. They should be able to use not fully known frameworks at a level enough to be sure of proper operating of the whole system.

1.1.3. **Software testing.** It is yet another of our experiences that students usually do not test their software. That happens because of

- wrong schedule not providing time for testing
- time delays, which consumes time for tests
- no knowledge about how to write certain test cases
- no knowledge on how to interpret certain tested system behavior

Thus, last but not least of the main goals is to force students to test their applications as much as possible. They are obligated to prepare and present their testing framework and executed test cases.

1.2. **Realization.**

1.2.1. **Teams and time.** Students are divided into teams of four persons. Time of the project is highly limited – 15 meetings of 6 hours. Students are not allowed to develop the project outside the labs. This creates environment similar to one in industry, where employees do not work at home but in offices (except for some home-working advanced and experienced employees). However students may organize time according to their needs and abilities. They may make breaks and perform brain-storm meeting in another separated room. Each team gets two or three computers and must organize schedule for all persons in a team to use resources as well as possible.

1.2.2. **Task.** To achieve goals presented above, we prepare a single project, which all teams must develop from an analysis up to a deployment. The task is usually some simulation application. The system must read simulation describing data in and then perform the simulation according to certain user selectable conditions. Already performed projects were (see appendices) sky scraper lifts simulation and car-flow junction simulation. These tasks were chosen because:

- topics are completely new to students
- allow complicated input data
- are big enough to be developed in a team
- may integrate several technologies (graphics, parsers, web, etc...)
- use a complicated engine and have to be well designed

1.2.3. **Methodology.** Students are allowed to choose their own developing and designing methodology. However, they are obligated to follow it strictly. For example they can start with OMT (Object Modeling Technique) and then use a waterfall software development model. Every week they are controlled if they follow the chosen method. For example if they choose XP (Extreme Programming), they must integrate every day and code stored in a repository must run, at least somehow. The educator always checks if they follow one chosen path.

A schema of overall organization is showed in the figure[1]. Each team (of several possible types) develops one task and is controlled by the educator. The task must integrate several technologies.
2. Education

An educator spends all the time with students. It is necessary in case they have problems or questions. At the end of the day teams upload their current project state (all artifacts) on a server. The projects are then analyzed by the educator on the next day.

2.1. Everyday control. The whole idea of the education is to give students clues which they usually do not get in classical studying process. They must be instructed on

- development process
  - how to organize the project
  - how to control the progress
- efficient team work
  - how to organize and share the code repository
  - how to organize document flow
  - how not to break working system
- assuring reliability
  - how to organize tests
  - how to maintain the development environment
  - how to refactor

We believe that in most cases the only way to help students in achieving these goals are practical laboratories joined with individual consultation repeated periodically. An educator analyzes the situation and helps students if such help is needed. These process must be performed all the time when students work. Final remarks on the end of the development process, which are usually given at standard labs are much too late. We believe that guided students will learn much more if they try to follow one required path from the beginning.

Thus, because of presented reasons we organize the educator and students interaction in a kind of everyday control. The one day activity of educator is showed in a figure 2.

2.2. Tools. Students may choose their own tools but are highly encouraged to use object-oriented CASE tools. Since professional systems are too expensive for our academia we use mostly free open-source applications and libraries: Poseidon UML, JUnit, JavaCC, CVS, etc. Documentation is usually created in HTML.

2.3. Judgment. Finished students’ applications are judged in several aspects (seem to be standard for student projects):

- quality of development process
- schedule control
- quality of testing
- keeping methodology
- quality of the finished product
- quality of documentation

All of the above points influence the final mark for each team. All teams should always finish their projects on time, since they are controlled all the time by the educator. However if they make serious mistakes or do not follow his clues they may run in troubles. Teams which do not finish on time get penalty points.

3. Conclusions

Our Software Development Laboratory is designed to answer needs of industry to get employees being able to organize a small development team and being open for new technologies. Constant development process control and selected tasks assure that students get the idea how a good team should work. We insist on schedule control and reliability of produced software.
Appendix A. Figures

Figure 1. The overall organization of teams, educator and tasks. Each team (of several possible types) develops one task and is controlled by the educator. The task may integrate several technologies.

Figure 2. The one day activity for the educator. Should be repeated for each team.