Experiences in teaching team software design

Pierre N. Robbillard

*Ecole Polytechnique de Montréal
Canada*

ABSTRACT

To teach team software design four components are needed: a team, a project, a process and quality control or evaluation mechanisms. This paper recounts our experience with a software engineering studio, the objective of which is that a team of students learns to design software and then to implement this software into a working product. Data gathered on team activity and the delivered products are presented. Comments and recommendations are made for each component.

Main conference themes: informatics as study topic

Educational areas: vocational education

Study topics: computer science/informatics

Secondary keywords: software engineering, teaching materials, teaching methods

INTRODUCTION

This paper recounts our experience with a course on practical software design called 'Studio on Software Engineering' (Atelier de Génie Logiciel IF-501). The Team, which is composed solely of students, realizes a software Project within a defined software environment Process. Quality control and evaluation are carried out continuously by the students and the instructors. The goal is to understand the role of each component and to improve its contribution to the learning process of the students.

The studio in software engineering is a half-year, 3-credit course offered in the fourth year of the computer engineering program. A prerequisite in software engineering, to be followed in the previous term, presents the concepts, and includes six practical assignments which serve as an introduction to the commercial software tools used in the studio.

Students work in teams of six, called companies. Each team or company is divided into three departments of two students each. Course enrolment is limited to 18 students. The students have a weekly one-hour meeting with a professional instructor knowledgeable in the problem domain and a full afternoon lab period with an instructor knowledgeable in CASE and process domains. Personal, technical and managerial issues of team components are discussed, and care is taken to reduce failure factors [1].

The studio aims at design and realization of a software module embedded in a much larger existing system. The module's input and output are predefined and no interface design is involved. The complexity and size of the modules to be implemented prevents any student from solving it on his or her own, and each module requires rigorous analysis.

Table 1 Software tools

<i>Tool Name</i>	<i>Purpose</i>
COSTAR™	Cost estimation
Microsoft Project Manager	Project planning & tracking
Software Through Pictures	Architectural design
SCHEMACODE™	Detailed design and coding editor
Compiler and Debugger	C compiling and debugging
DATRIX™	Quality assessment

Each team applies a defined process based on systematic reviews at each stage of the software development process. The software process provides the guidelines for team interaction, project monitoring and quality assessment [2] and is derived and adapted from an industrial process. This process integrates

the tools listed in Table 1. The students receive training in technical reviewing, programming standards and quality assessment. Also videos on formal inspections and scenarios for software inspections are presented [3, 4, 5].

The studio clearly distinguishes between quality control and personal evaluation. Quality control activities are not marked. Quality control is performed by the students themselves with the help of a measuring tool. Formal acceptance tests are conducted at delivery time by the instructor.

Team components

Simulating teamwork with students is not a trivial issue and there is no easy way to go about it. The dynamics of building student teams is determined by a class of students evolving into a group and finally into a team. A class is a body of students meeting regularly to study the same subject. Here a student is an individual who does not need to exchange information with others. A group is a number of individuals working on a group assignment. Here a student is linked to the group assignment, not to the group activity. A team is a number of persons associated in an activity, i.e. linked into the activity of the team.

There are three factors relating to an individual's attitude:

1. Motivation can be an issue with class or group members, but hardly with team members. Marks are seen as a quality evaluation of the team's product, and not as an evaluation of personal performance because all team members get the same mark.
2. Uneven work load is minimized by the project structure. There is only a certain amount of work which can be done for a given task. The students are regularly reminded to consider product or process compromises instead of overtime resources.
3. Foreseeing a problem before it is too late is important. The 'fantasy factor' and the students' inability to estimate task effort, or their lack of experience in doing so, play a role.

Team members all should be at the same technical level, otherwise shared activities will result in frustration. Here also three factors are relevant:

1. Programming experience is minimal. A programming guide is given to the students to improve programming practices and product uniformity.
2. The fantasy factor comes into play here when designers or programmers want to add features of their own. Such well-intentioned last-minute modifications are the most difficult to control because they originate from a legitimate desire to improve the product. These may well lead to failure of the project because of the inability of the students to anticipate the work load involved.

3. The version or variation control factor is important in any teamwork situation. We experienced one project failure due to last-minute changes. Version control should be automated and be under the responsibility of an assigned team member.

Management is influenced by three factors which are related to team structure:

1. Students are free to join any team. Based on the previous definitions, we had one team and two groups each year. The teams were composed of students who wanted to work together before entering the course. The groups were composed of students who wanted to take the course, but did not know in advance with whom they would work. It is very difficult to make up a team from students who do not know each other beforehand. There is just not enough time.
2. Coordination of group meetings is an issue. No formal project organization structure is followed, although normally a team leader emerges in each team. We strongly support the concept of selfmanaged work teams [9]. With this strategy, management determines what is to be done and the team decides how it will be done. Students commented that they were not ready for this type of management structure and asked for formal sessions on team management.
3. Ego (the I am right feeling), was reduced by each department (of two people) changing jobs for every task.

Project components

The studio was to run for two consecutive years with a different project each year. The project structure was the same and had the following characteristics:

- functionality embedded within a large system;
- no interface design and implementation;
- naturally divided into three subprojects or tasks (reasons explained below);
- application unfamiliar to the students (no games or the like);
- expected realization time, 810 hours.

The first year project produced a module of a re-engineering tool which had to restore structure to the graph of the flow of control and transform the resulting graph into pseudocode. The second year project produced a module of a software measuring tool for linking files, doing some sorting and computing metrics on resulting files. The students were given all the necessary theoretical information, documentation, tools and input files.

Table 2 Work alternation for company

<i>Phase/subproject</i>	<i>Task 1</i>	<i>Task 2</i>	<i>Task 3</i>
Planning	A	B	C
Analysis	C	A	B
Realization	B	C	A
QA & Integration	A	B	C

The tasks were distributed among the team members according to a predefined scheme. A team is a company with three departments. Each project was divided into three tasks, one for each department. Within each company, the departments worked alternately on each task, as shown in Table 2. Each department does the quality assessment and integration (QA & Integration) for the task they first planned. This approach proved to be useful in reducing ego programming and justifying the need for proper documentation at every phase.

Software process components

Software design is a step-by-step activity which transforms abstract specifications into real source code. A creative designer makes one-of-a-kind software which is a personal chef d'oeuvre.

Table 3 Process elements

<i>Phase</i>	<i>Abr.</i>	<i>Document</i>	<i>Tool</i>
Specification	SP	Specification	Frame Maker
	SW	Specification walk-through	Frame Maker
Preliminary design	DA	Architectural design	Software through Pictures
	PT	Task description	Frame Maker
	PE	Project planning	Microsoft Project 3.0
	PW	Planning walk-through	Frame Maker
Detailed design	AT	Detailed Task analysis	SCHEMACODE
	DF	Design diagrams	Software through Pictures
	AW	Analysis walk-through	Frame Maker
	RT	Implementation	SCHEMACODE
Realization	IC	Source Code Inspection	SCHEMACODE
	RD	Quality assessment	DATRIX
	RW	Realization walk-through	Frame Maker
	TC	Unit-level test case	SCHEMACODE
Integration	IT	Task integration	SCHEMACODE
	IM	Integration walk-through	Frame Maker

Software engineers design reliable, quality controlled software. Software organizations are evaluated nowadays in terms of the maturity of their software process. A comprehensive software process exceeds the scale of the studio project. A downsized software process adapted to the studio environment is presented in this section. Table 3 defines the documents produced and the corresponding software tool used in each phase. Figure 1 defines the process and corresponding documents according to the definitions in Table 3. This iterative process flows from the specification activity to the integration activity. The information contained in each textual and review document is based on IEEE software engineering standards [6].

The specification (SP) document describes functionality and theoretical background, and is written and reviewed by professional software engineers.

Fig. 1 Software process

Students run the Specification Walk-through (SW) to make sure they understand it. Then they produce a set of task descriptions (PT) during the preliminary design phase. Schedules (PE) are set up according to the resources available and the established milestones. Software through Pictures (StP) is used to draw the basis of the architectural design (DA). The specification and preliminary design phases are carried out at the project level. The following phases are carried out for each of the three project tasks. The students use StP during the detailed design phase to produce the various design diagrams: data flow, data structures and hierarchical diagrams. They use the structure based editor SCHEMACODE to describe the function algorithms (AT).

The realization phase includes detailed design, coding, quality assessment and source code inspection. The detailed design is completed with SCHEMACODE, which generates C source code from the schematic pseudocode (RT) [7]. Schematic pseudocode is an efficient code inspection (IC) tool. DATRIX processes the source code to ensure that it conforms with the company's metrics profile. Each function outside the profile has to be explained or rewritten (RD) [8]. The last phase is testing (TC) and system integration of the functions (IT). A final product walk-through is conducted before the competition.

All students must rigorously follow this process and documentation templates are provided. A schedule is planned for handing in documents, and all documents are marked.

Quality control and evaluation components

Quality control is an assessment of student work which is not marked, while an evaluation is an assessment of student work which contributes to the team's final mark. Quality control of the product is performed by the students using a static analyzer. Programs have to meet a static metrics profile [8]. The instructor evaluates the process documentation and the deliverables through acceptance tests.

The process documentation is evaluated according to the following criteria: completeness, and forward and backward tracing. Any software unit documentation must be linked backward to a corresponding specification requirement, and, inversely, all specification requirements must be linked forward to their source code implementation. All documents are built from existing templates.

Quality control of the software product is performed by a software tool called DATRIX. The static analyzer parses all source code and computes 30 metrics. A profile or a usual range value is given for each metric. Students must verify that metrics for all software units fall within the given range. Unusual metric values must be explained.

Evaluation of the software product is based on the number of successful acceptance tests. All products are evaluated on the same day by the instructor. Each team is required to deliver in time. The fifteen (15) acceptance tests are straightforward and designed to evaluate most of the functionality of the product.

Data and comments

This section presents data collected during this year's studio and discusses some of the results. All data, except time log data, are collected automatically.

Each student fills out a weekly time-log recording the time spent on the activity. Table 4 integrates the data for every team and relevant activity types. The total time in Table 4 accounts for approximately 75% of the time recorded.

Table 4 Time spent by teams on each activity

<i>Activity Team</i>	<i>A</i>	<i>A%</i>	<i>B</i>	<i>B%</i>	<i>C</i>	<i>C%</i>
Preliminary Design	182	20	62	12	104	29
Detailed Design	342	40	183	37	116	32
Implementation	160	18	142	29	104	29
Integration	197	22	113	22	37	10
Total Time (hr.)	881		500		361	

Each team officially had 810 hr. to spend on the project (15 weeks x 9 hr. x 6 team members). Team A had real team spirit, as defined previously, while teams B and C had more of the group style. Team A was enthusiastic and exceeded the time allowed. They spent more time on the design phases than the others who spent around 50% of their time on design activities. Team (group) C found in the implementation phase that the product was unlikely to run. Little effort was therefore put into integration. The percentage of time spent on each phase is more or less fixed by the process milestones. Group B spent less time on the preliminary design because they did not start on the project right away (during the first week).

Table 5 presents data on the delivered products. The two program size metrics were computed automatically: number of statements and number of software units. 'Documented software units (S/W)' gives the percentage of documented units. 'Documentation content' is a percentage representing the average evaluation of the five documents (PT, DA, AT, DF, RT) by the instructor on the criteria accuracy and completeness. 'Acceptance test' gives the percentage of successful tests. Programs were subjected to 15 tests. Team A was successful in 12 tests, team B passed 9 tests and team C did not complete the integration phase.

Team A realized the best product. It is small, complete and well documented. Group B did a fair job. Only 18 of the 33 software units (55%) were documented. Members of Group C had an individual approach to software development and did a poor job.

Table 5 Data on the delivered products

<i>Items / Teams</i>	<i>A</i>	<i>B</i>	<i>C</i>
Statements	2484	3989	4128
Software units	27	33	67
Documented S/W	100%	55%	20%
Documentation content	86%	80%	66%
Acceptance tests	80%	60%	0 %
Overall rating	very good	fair	poor

CONCLUSION AND RECOMMENDATIONS

We believe that team cohesion is a key component which is difficult to control since students meet for only few hours a week. We find that teams whose members already know each other, perform better. Differences between a team and a group are significant and marks should be weighted accordingly. Objective measures for team cohesion are, however, lacking. A lecture on team cohesion by a professional is expected to be well received by the students.

Problem solving and application design should not be confused. A design oriented course should be based on well-defined and easy-to-understand applications. Fantasy factors should be identifiable and not permitted.

Students should already be familiar with all the software tools used. The learning curve for a new tool can be significant.

It is utopia to simulate an industrial environment in a classroom. Practical internships are available to students who want full-time experience in the industrial milieu. The project and the process should be downsized and adapted to the proper environment.

Quality control provides the guidelines for the project. Students need established limits, since in some cases they do not have enough experience to judge for themselves. The quality criteria should be well defined and reachable by the average team member. Objective of the course should be to show that good design is feasible.

ACKNOWLEDGEMENTS

We thank the members of our group, with special thanks to André Beaucage, and to Jean Mayrand, Jean-Sébastien Neveu and Philippe Mathieu who acted as instructors during the past two years. We extend special thanks to Jean Normand Drouin, professional evaluator from Bell Canada who assessed team processes. Support for this work was provided in part by the Service pédagogique de l'Ecole Polytechnique and BELL CANADA.

REFERENCES

1. Kantipudi M., Collier, K.W., Collofewllo, J.S. and Medeiros S. (1992) *Software engineering course project: failures and recommendations*. Software engineering education, SEI Conference, San Diego, Springer Verlag, pp. 324-338.
2. Robillard, P. N., Mayrand, J. and Drouin, J.N. (1994) *Process Self-Assessment in an Educational Context*. Software engineering education, SEI Conference, San Antonio, Springer Verlag, pp. 211-225
3. *Software Engineering Curriculum Project*, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA 15213-3890
4. Freedman, D.P. and Weinberg, G.N., (1990) *Handbook of Walkthroughs, Inspections, and Technical Reviews*, Third Edition, Dorset House, NY.
5. Oualine, S., (1992) *C Elements of Style: the Programmers Guide to Developing Well-Written C and C++*. M&T Books.
6. IEEE Software Engineering Standards collection, Spring Edition, NY, 1991.
7. Robillard, P.N. (1986) *Schematic Pseudocode for program constructs and its computer automation by SCHEMACODE*. Communications of the ACM **29** (11) pp. 1072-1089.
8. Robillard, P.N., Coupal D. and Coallier F. (1991) *Profiling Software through the Use of Metrics*. Software - Practice and Experience **21** (5) pp. 507-518.

9. Schlimmer, J. C., Fletcher, J. B., and Hermens, L. A. (1994) *Team-Oriented Software Practicum*. IEEE Transaction on Education **37** (2) pp. 212-217.

COSTAR™ is a registered trademark of Sofstar.

DATRIX™ is a registered trademark of Bell Canada.

SCHEMACODE™ is a registered trademark of Schemacode International Inc.

Software Through Pictures™ is a registered trademark of Interactive Development Environments.