Relatório Técnico

Núcleo de Computação Eletrônica

Teaching Communication and Interpersonal Skills in an Extreme Programming Course

V. M. Teles
C. E. T. de Oliveira

NCE -25/02

Universidade Federal do Rio de Janeiro

Teaching Communication and Interpersonal Skills in an Extreme Programming Course

Vinícius Manhães Teles
Universidade Federal do Rio de Janeiro
Núcleo de Computação Eletrônica
vinicius@improveit.com.br

Carlo Emmanoel Tolla de Oliveira
Universidade Federal do Rio de Janeiro
Núcleo de Computação Eletrônica
carlo@ufrj.br

Abstract

The major problems of software development projects are not so much technical as sociological in nature. The industry seems to agree very much with this statement while the university seems to give it little importance. This article analyzes the social problems that affect software development teams with special attention to communication. It shows that universities and industry have different views about what skills students should develop. The article proposes the introduction of an extreme programming course which can address some of the main concerns expressed by industry.

1. Introduction

The main emphasis in Computer Science courses is on the development of technical skills by the students undertaking the course (IEEE/ACM apud Pham, 97).

This focus on technical skills poses problems for the future professionals when they are supposed to build information systems where their activities are part of a team effort. According to DeMarco et al. [1] since they work in teams and projects and other tightly knit working groups, they are mostly in the human communication business. Their successes stem from good human interactions by all participants in the effort.

Researches conduced in Australia seem to confirm the statement above. In this country, surveys of employers have shown that the qualities they consistently rate most highly in graduates relate to their communication skills, their ability to work together in teams and their technical writing skills, besides their basic technical knowledge. (Business Higher Education Round Table and Department of Employment, Education and Training apud Keen, 98)

The Business/Higher Education Round Table, Australia conducted a survey in 1992 in which both business and universities were asked to rank the desired characteristics of university graduates. The results are shown in Table 1 [2].

Table 1. Desired Characteristics of University Graduates (where 1 is the most desired)

Desired Characteristics of Graduates	Rank Business	University
Communication skills	1	7
Capacity to learn new skills and procedures	2	5
Capacity for cooperation and teamwork	3	8
Capacity to make decision and solve problems	4	3
Ability to apply knowledge to workplace	5	4
Capacity to work with minimum supervision	6	6
Theoretical knowledge in a professional field	7	1
Capacity to use computer technology	8	2
Understanding of business ethics	9	12
General business knowledge	10	11
Special work skills	11	9
A broad background of general knowledge	12	10

This survey indicates that business and universities differ in their ranking of the importance of characteristics in the graduates in two major areas:

1. Communication skills, a capacity to learn new skills and procedures, and a capacity for cooperation are teamwork. In each of these cases the universities rankings and well below those of business, particularly in the area of communication skills.

2. Theoretical knowledge in a professional field and a capacity to use computer technology. In these cases universities have rated these characteristics much higher than has business. [2]

These results are consistent with the views of some authors, like DeMarco and Lister [1], Goguen and Linde [14] and Goguen [8]. They believe that software development is strongly affected by social issues. So, it cannot be treated in a purely technological way.

Some researchers have been studying methods to improve the education of software engineering in order to address these social issues. In this paper, the authors propose a semester long course on Extreme Programming which uses the results of recent works in the field. They describe some of the characteristics of this methodology and how it addresses some of the problems described so far when taught using the techniques described ahead.

2. Extreme Programming

In the teaching of software engineering, emphasis is on the procedures of software development. Software development is comprised of requirements elicitation and analysis and the design, development and implementation of the system. Technical skills are required to produce a working system and management skills are required to effectively handle the process. However, software development activities require additional underlying skills – communication skills, problem-solving skills and the ability to work as a member of a team. [10]

Extreme Programming (XP) has a special approach to deal with the steps required to develop a software system. In particular, it gives much attention to communication and teamwork skills. It began to be developed in 1996, Smalltalk by code developer and consultant Kent Beck, with authors Ward Cunningham and Ron Jeffries. XP is now practiced by programmers worldwide. XP methodology's success rate is so impressive that it has aroused the curiosity of many software engineering researchers and consultants. [4]

Extreme Programming (XP) is a lightweight software development process for small teams dealing with vague or rapidly changing requirements [5]. It defines a set of values and principles that must be followed by project team members. According to Beck [3, p.29] the values are:

- Communication
- Simplicity
- Feedback
- Courage

The principles [3, p.37] are:

- Rapid feedback
- Assume simplicity
- Incremental change
- Embracing change
- Quality work

Jeffries et al. also describe a set of practices in [6] which can be summarized as follow:

- On-site customer
- User stories
- Acceptance tests
- Unit tests
- Story Estimation
- Small Releases
- Iteration Planning
- Quick design
- Refactoring

For the purposes of this work the authors focus on the values of communication and feedback. And also give special attention to the practices: on-site customer and pair-programming.

2.1. Communication

The first value of XP is communication. Problems with projects can invariably be traced back to somebody not talking to somebody else about something important. Sometimes a programmer doesn't tell someone else about a critical change in the design. Sometimes a programmer doesn't ask the customer the right question, so a critical domain decision is blown. Sometimes a manager doesn't ask a programmer the right question, and the project progress is misreported.

Bad communication doesn't happen by chance. There are many circumstances that lead to a breakdown in communications. A programmer tells the manager bad news, and the manager punishes the programmer. A customer tells the programmer something important, and the programmer seems to ignore the information.

XP aims to keep the right communications flowing by employing many practices that can't be done without communicating [3, p. 29].

Communication is a fundamental value, since according to Jeffries et al. [6], successful software development is a team effort – not just the development team but the larger team consisting of customers, management, and developers. Extreme Programming is a simple process that brings these people together and helps them to succeed together.

Gouguen [8] also express his concerns on the subject of human interaction present on software development projects and affected directly by communication. According to him, much of the information that requirements engineers need is embedded in the social worlds of users and managers, and is extracted through interactions with these people, e.g., through interviews and questionnaires. At its source, this information tends

to be informal and highly dependent on its social context for interpretation.

2.2. Pair Programming

Pair programming is a style of programming in which *two* programmers work side-by-side at *one* computer, continuously collaborating on the same design, algorithm, code or test [7].

In pair programming, two programmers jointly produce one artifact (design, algorithm, code). The two programmers are like a unified, intelligent organism working with one mind, responsible for every aspect of this artifact. One partner, the driver, controls the pencil, mouse, or keyboard and writes the code. The other partner continuously and actively observes the driver's work, watching for defects, thinking of alternatives, looking up resources, and considering strategic implications. The partners deliberately switch roles periodically. Both are equal, active participants in the process at all times and wholly share the ownership of the work product, whether it is a morning's effort or an entire project [4].

Pair programming inherently incorporates basic design and review phases in the development process. The programmers must communicate about possible approaches in order to discuss appropriateness during development. Both members review the other's input, a process that increases the developers' ability to perform effective reviews [11].

2.3. Feedback and customer on site

Feedback is one of the most important characteristics of an extreme programming project and one of the main reasons why these projects use to progress faster. The best way to get rapid feedback is to have the customer available as much as possible.

XP recommends the customer to join the team throughout the entire project. This way, the team can have feedback fast and can fix things early [3, 6].

3. Teaching Techniques

3.1. Open Ended Group Projects

To teach extreme programming and its characteristics, the authors will use the idea of open ended project groups (OEGP). It is a form of experimental learning (Kolb apud Daniels et al., 02) which can, in principle, be used to advantage to teach any subject with a practical application [9].

Daniels et al. [9], state that in addition to supporting knowledge acquisition, OEGP can be used to help the students gain and improve skills. The most obvious skill areas which are involved are interpersonal communication and group working. However, a suitably designed OEGP can ensure that students must consider the problems of communication with manager and client and can help improve both report writing and presentation skills. OEGP also assist in getting students to analyze problems and synthesize solutions while examining, and trying to mitigate the risks of things going wrong, all valuable skills for the software engineering project managers of the future.

The experience of group project work prepares the students for their subsequent careers where group working is the norm. Undertaking open ended projects also appears to have the benefit that they force the students to think about the problem rather than spending time searching for the 'correct' answer.

It is also noted that OEGP appear to have measurable beneficial effects on student performance in other academic subjects. Improved motivation and greater enthusiasm seem to carry over into general performance, confidence levels go up and problem solving skills improve so that students are more willing to attempt difficult tasks. OEGP can also be used to encourage students to apply theory which should lead to a better understanding of the theory and thus to improved performance in examinations.

Daniels et al. [9] describe several projects run in the classroom and they describe the students' reaction. Firstly, feedback from the students has been generally very positive. In all of the OEGP with which the authors have been involved there has been positive feedback from the students both during the module and afterwards. It is also noticeable that the levels of motivation of the students appeared to be higher with better completion rates, less plagiarism and very few drop outs or failures.

But OEGPs raise some criticism. Daniels et al. [9] explains that the main concerns that are expressed when the use of OEGP is suggested relate either to the use of group projects at all ("weak students get 'carried', good students get 'pulled down'") or to the fact that the outcome for an OEGP is inherently unknown i.e. that there is no 'right' answer. The necessity for group working is, however, becoming more widely accepted now (Ford apud Daniels et al., 02), thus it is the concerns about the open ended nature of the project and the need for fair assessment based on problems for which there is a correct answer, which are addressed here.

The obvious counter argument is that OEGP mirror real life software engineering projects which do not usually have known 'right' answers and there is a need to assist students to learn this before they start to work. Part of this learning process includes the intrinsically difficult process of finding out what the client thinks is required (Veryard apud Daniels et al., 02), negotiating with the

client to agree what can be done and, later, explain what has actually been done and how it relates to the requirements. An alternative argument is that, ultimately, all criteria are established and judged by people and are, therefore, subjective. Objective criteria are only regarded as objective because there is agreement about the 'correct' way in which something should be done, or said. History suggests that most such agreements change over time and current 'right' way may well be revised later.

A different perspective on the fair assessment of OEGP can be provided by considering the way in which science and engineering are advanced. All research projects have unknown outcomes but the methods used to undertake and present research are common. Thus it is possible to provide a fair assessment process for OEGP by focusing on the process which the students use rather than the product they produce [9].

3.2. Motivation and ice-breaking

Ratcliffe et al. [15] describe the adoption of an integral activity weekend as part of the faculty's introductory software engineering course and a second weekend to reinforce industrial awareness in the student's second year.

These weekends have been developed to improve motivation and staff-student relations, emphasizing on life skills and adaptability. The idea is to introduce students to the concept of team skills. With specific attention to personal challenge and team dynamics, these weekends where carefully designed to both improve motivation and enhance the general employability of the students.

They follow a series of specially tailored outdoor activities that are designed to promote self and inter-personal skills through a series of shared group experiences. The activities are personally challenging and are heavily teamwork oriented. They cannot be carried out successfully without group co-operation and group encouragement.

Ratcliffe et al. [15] state that the response from the weekends is overwhelming. The students obviously enjoy themselves a great deal but more importantly they learn a great deal about themselves and working with others. It seems that the students learn far more about team working in one weekend than could have been taught to them through a whole series of class projects.

Although the approach is very interesting and effective, the costs are high. Considering the limitation of resources that affect many universities, it's necessary to find a way to implement these weekends with lower costs.

3.3. Large Scale Projects

Sebern [16] works with the idea of large-scale software development. It proposes a software development laboratory in which teams work for extended periods on large-scale, ongoing projects in the context of a standardized and evolving development process. It's composed of a three-course sequence.

Students in this course believe they have achieved the objectives related to teamwork, process improvement and software development practice.

This sequence aims to reflect much of the experience in the "real-world". But it is hard to be implemented by faculty staff.

4. Proposal

The authors propose a semester long course for a class composed of 20 students. The content is based on the books of Beck [3], Beck and Fowler [13] and Jeffries et al. [7].

The introduction of extreme programming teaching to computer science undergraduate courses is not new. It has already been described in the works of Shukla and Williams [12] and Müller and Tichy [5].

Shukla and Williams [12] describe a course based on a 16-week semester class where the students completed four Java programming projects during the course of the semester. Three of the projects were completed as the students were learning and using more traditional software development practices. These practices were based on the Collaborative Software ProcessSM (CSPSM) developed by Williams.

They found that one semester is not long enough to teach two very different methodologies nor for the students to perform meaningful assignments using two very different methodologies.

Furthermore, the students didn't work at all times co-located and with a customer on site. So, the experience didn't reflect the work of a real extreme programming project where co-location is extremely recommended and the customer should be present.

Müller and Tichy [5] developed an extreme programming course where, in the first three weeks students solved small programming exercises to familiarize themselves with the programming environment and to learn XP practices. The exercises introduced jUnit (the testing framework used throughout the course), pair programming, the test practices of XP (write test cases before coding, execute them automatically with jUnit), and refactoring. The remaining eight weeks were devoted to a project on visual traffic simulation. The course language was Java. All students had experience with Java from their early undergraduate courses.

As in the case described by Williams, the students didn't work at all times co-located and with a customer on site which brought some problems.

The proposal of this paper tries to overcome the problems found in both cases. Firstly, it focuses only in one methodology, the extreme programming. Secondly, the students will work only in one project through the course. And they will develop software during the classes, so they will always be co-located and will always have the customer on-site.

Real world problems with real clients seem, initially, to be ideal. The real clients can explain the problem and interact with the students. However, this ignores the educational issues, which may also result in very dissatisfied clients. The client has a problem and would like a solution. The students are undertaking the task of solving the problem, not primarily to produce a solution but to learn how to solve problems like this and to gain academic credit for their attempts.

Research projects can be an excellent way of offering the benefits of having real clients who have realistic expectations of the likely outcomes. However, it is still necessary to recognize that the educational outcomes must take precedence for the students over the research objectives. [9]

This idea will be used in this course. The project will be the development of software to support the dissertation of a graduate student who will act as the customer all over the course. This approach is interesting, because the teacher won't act as the customer. He will act as the coach of the team. This project will use Java as its development language.

The class will be divided in two to form two teams of 10 students each. In each team, the students will always work in pairs. They will practice pair-programming at all times following the recommendations of Williams et at. [4][7].

The students will be evaluated in two ways, according to their individual achievements and their behavior working in the team.

Every week the teacher will propose a reading assignment for the students. They will have to read a text and answer the questions posed by the teacher in the next class. The texts are used so that students can learn more about each characteristic of the methodology. This evaluation will represent half of the week grade.

The other half is based on teamwork. Students will be observed while they work in pairs and the teacher will grant the marks according to the way they communicate with the pair and the other teammates.

The teacher won't evaluate the final product, but only the process in which students build the product. So, the student's are not supposed to work on the project when they are not in class.

These are 4 hour classes once a week. In the first hour the teacher will ask the students about the text they've read along the week before the class. In the second hour the teacher will give a lecture on the subject of the class. And in the remaining two hours the students will work on the project.

The authors expect to overcome the problems found in previous experiments of this type. And hope the students will improve their communication and teamwork skills.

5. Conclusion and Future Work

This paper began describing the concerns of the industry with graduate standards in the areas of communication and interpersonal skills. Both empirical research and anecdotal evidence confirms that industry remains strongly concerned over the teaching of this area.

The authors suggest the introduction of and extreme programming course to address some of these concerns. They also present some characteristics of extreme programming and previous experiences of this type. Finally, they describe the structure of the proposed course which is already underway at Federal University of Rio de Janeiro.

This experiment is the first of a series of experiments the authors intend to put in action in their research which looks for new ways of teaching software engineering in order to shorten the gap between industry expectation and university expectation. The results of this experiment will be available in the future in the form of a new paper.

References

[1] T. DeMarco, T. Lister, *Peopleware Productive Projects and Teams*, Dorset House Publishing Co., New York:, 1987.

[2] C. Keen, C. Lockwood, J. Lamp, "A Client-focused, Team-of-Teams Approach to Software Development Projects", *Proceedings of Software Engineering Education & Practice (SE: E&P '98)*, IEEE Computer Society Press Dunedin, 34-41

[3] K. Beck, *Extreme Programming Explained – Embrace Change*, Addison Wesley, 2000.

[4] L. Williams, R.Kessler, W. Cunningham, R. Jeffries, "Strengthening the Case for Pair Programming", *IEEE Software*, vol. 17, pp. 19-25, July 2000.

[5] M. Müller, W. Tichy, "Case Study: Extreme Programming in a University Environment", *Proceedings of the International Conference on Software Engineering 2001 (ICSE 2001)*.

[6] R. Jeffries, A. Anderson, C. Hendrickson, "Extreme Programming Installed", Addison-Wesley, 2001.

[7] L. Williams, R. Kessler, "All I Really Need to Know about Pair Programming I learned in Kindergarten," *Communications of the ACM*, vol. 43, pp. 108-114, May 2000.

[8] J. Gouguen, "Requirements Engineering as the Reconciliation of Technical and Social Issues", in Requirements Engineering: Social and Technical Issues, edited with Marina Jirotka, Academic Press, 1994, pp. 165-199.

[9] M. Daniels, X. Faulkner, I. Newman, "Open Ended Groups Projects, Motivating Students and Preparing them for the 'Real World'", *Proceedings of the 15th Conference on Software Engineering Education and Training (CSEET'02)*.

[10] A. Goold, P. Horan, "Foundation Software Engineering Practices for Capstone Projects and Beyond", *Proceedings of the 15th Conference on Software Engineering Education and Training (CSEET'02)*.

[11] J. Bevan, L. Werner, C. McDowell, "Guidelines for the Use of Pair Programming in a Freshman Programming Class", *Proceedings of the 15th Conference on Software Engineering Education and Training (CSEET'02)*.

[12] A. Shukla, L. Williams, "Adapting Extreme Programming for a Core Software Engineering Course", *Proceedings of the 15th Conference on Software Engineering Education and Training (CSEET'02)*.

[13] K. Beck, M. Fowler, *Planning Extreme Programming*, Addison Wesley, 2001.

[14] J. A. Goguen, C. Linde, "Techniques for Requirements Elicitation", *Proceedings of Requirements Engineering (RE'98), IEEE Computer Society*, 1998, pp. 152-164.

[15] M. Ratcliffe, J. Woodbury, L. Thomas, "Improveint Motivation and Performance Through Personal Development in Large Introductory Software Engineering Courses", *Proceedings of the 15th Conference on Software Engineering Education and Training (CSEET'02)*.

[16] M. J. Sebern, "The Software Development Laboratory: Incorporating Industrial Practice in an Academic Environment", *Proceedings of the 15th Conference on Software Engineering Education and Training (CSEET'02)*.

[17] B. Pham. "The changing curriculum of computing and information technology in Australia". In *2nd Australasian SIGCSE Conference*, pages 149-154, The University of Melbourne, July, 1997.