TEACHING SOFTWARE ENGINEERING THROUGH COLLABORATIVE METHODS

Dr. Alan R. Peslak, Penn State University, arp14@psu.edu

ABSTRACT

Engineering of Complex Software Systems (IST 412) is a senior level software engineering course in the Penn State University Information Sciences and Technology program. This paper outlines some of the collaborative techniques and approaches used in the course to simulate as closely as possible the multiple stakeholders and teams that are an essential parts of software development today.

Keywords: Software engineering, collaborative methods, team projects, agile methods

INTRODUCTION

Software Engineering (SE) by its very nature is a collaborative process. Complex software systems nearly always have teams of analysts and programmers developing the software. In addition there are multiple stakeholders in any SE project. These stakeholders range from users to managers, from external agencies such as the government to internal multiple functional departments within an organization. Many of the traditional methods of instruction for information systems and computer science such as lecture, brief assignments, and papers, give no appreciation of the need for collaborative efforts in real world SE projects.

In a senior level IST 412 course, The Engineering of Complex Software Systems, the author attempted to develop a program that incorporated multiple collaborative assignments that would simulate many of the issues that are inherent in real world collaborative software projects. Specifically the exercises dealt with a series of areas. These areas were:

- Requirements analysis,
- Pair programming,
- Coding standards,
- Needs assessment,
- Function points.

In addition, a major term project was included in the course. The project required a team effort in the development of a comprehensive plan document including major components of a complex software engineering project.

REVIEW OF LITERATURE SUPPORTING SE COLLABORATIVE METHODS

Group and collaborative projects in software engineering have been firmly established and widely adopted by educators. Some of the benefits noted by the Brereton et al. (2) include:
Providing development of more complex and more real world type systems,
Allowing team development and processes again simulating real world environments,
Providing an opportunity for students to learn from each other as well as share skills and knowledge,
Providing an opportunity to face and address team and group problems such as scheduling, time allocation, and agreement on requirements.

These are the factors which most often doom real world SE projects.

Ludewig and Reibing (4) emphasize the importance of application-oriented problems. Some of the changes to the software curriculum they have incorporated include:

- Application of knowledge experience,
- Academic software engineering projects with strict project management,
- Combination of knowledge or application areas with SE procedures.

The authors see the value of practical experience as well, but suggest that many companies do not follow SE procedures. There may be less to be gained from actual real world environments than expected.

Lethbridge (3) ranked 75 potential software engineering topics and asked software practitioners to rank them in order of importance. Among the top fifteen topics noted as most important were:

- Programming languages,
- Software design,
- Software architecture,
- Requirements gathering and analysis,
- Object-oriented concepts and technology,
- Analysis and design methods,
- Giving presentations,
- Project management,
- Design of algorithms,
- Technical writing.

The collaborative techniques taught in IST 412 emphasized many of these areas.

The Carnegie-Mellon Software Engineering Institute published guidelines for SE education (1) and proposed the following concepts be included in an SE curriculum:

1) Software engineering “in the small” – individual software development projects,
2) Software engineering “in the large” – team software development of a small to moderate sized “industrial” project,
3) Product activities – methods, techniques and skills to build software components, including requirements analysis, QA plans, design specifications, and documentation
4) Process activities – standards, guidelines, techniques, and plans, providing an SE framework.

The collaborative activities in IST 412 included the recommendations of steps 2 through 4.

2) A term, team project in an overall software development plan,
3) Specific techniques to aid in SE such as pair programming, function point analysis and requirements analysis,
4) Review and reference to numerous standards, templates, guidelines, and planning techniques to prepare their overall term software project.

SOFTWARE ENGINEERING COLLABORATIVE TERM PROJECT

The most significant collaborative effort in IST 412 was the major team-based term project that had as its deliverables a series of SE documents modeled after the Pfleeger text (5) including:

- Project scope,
- SDLC model,
- Maturity model,
- Project overview,
- Project plan,
- Feasibility analysis,
- Requirements analysis,
- Design,
- Coding,
- Testing,
- Maintenance,
- Delivery.

A prototype was not required for the project, but the project was expected to be at least 100 pages in length, suggesting a fairly comprehensive coverage of each major area.

Teams were formed at the start of the class. Team formation was made by the instructor. A general questionnaire was completed by each student on the second day of class. The questionnaire included questions on general availability schedule (outside of class) as well as self-rating of skill levels in presentation skills, writing, business experience, and software engineering experience. The instructor examined each document and attempted to group students with a mix of skills and similar availability schedules. One of the major problems with collaborative projects in the past has been the inability for students to meet outside of class due to conflicting work schedules. To the greatest extent possible, this issue was addressed.

The rubric used for grading the project at the end of the term is shown in Figure 1. The document was examined for inclusion of each general element and then graded on a scale of 1 to 5 as noted based on whether specific requirements were met as well as the quality of each item.
One of the teams developed a comprehensive 237 page document detailing all of the major processes involved in an SE project. Their high level table of contents is presented.

1.0 Problem Definition
2.0 Team Formation
3.0 SE Paradigm
4.0 Project Feasibility
5.0 Project Planning Methodology
6.0 Project Estimates
7.0 Project Schedule
8.0 Staff Organization
9.0 Tracking and Control Mechanisms
10.0 Requirements Analysis Process
11.0 Risk Analysis Process
12.0 Design Process and Standards and UML
13.0 Coding Process and Standards
14.0 Testing Process
15.0 Delivery and Acceptance Process
16.0 Maintenance Process
17.0 CMM
18.0 URL
19.0 UML
20.0 Data Dictionary
21.0 References

This was an impressive contribution from a four-person undergraduate team.
OTHER SOFTWARE ENGINEERING COLLABORATIVE METHODS UTILIZED

There were a series of other collaborative exercises used as in-class assignments. Some of the major exercises follow.

Requirements analysis

Students were grouped in teams of four and charged with the following task at the beginning of a 75 minute class period.

Perform a mock requirements analysis for a hospital portable patient record keeping system (PDA used by nurse, doctor, etc.) to record and retrieve patient information. You may use the model in Pfleeger or some other template.

Pair programming

Students were grouped in teams of two and charged with the following task at the beginning of a 75 minute class period.

Using a programming language of your choice (Visual Basic, Java, C++, Visual Basic .NET or other), develop a basic program that accepts input with a GUI front end, performs calculations or manipulations on the data entered and displays or prints information based on the input and calculations. One person performs the actual coding, the other observes and comments on the program and process. The roles may be switched during the 75 minutes. If you accomplish basic functionality, then add other functionality if possible in the 75 minute time frame. (6)

Coding standards

Students were grouped in teams of two and charged with the following task at the beginning of a 75 minute class period.

You have been given a series of Java classes. All of these classes are acceptable Java code and will compile and run. However there are at least 15 deviations in the code based on Java coding standards. You need to review the Java coding standards, available at http://www.sun.com through ftp://ftp.javasoft.com/docs/codeconv/CodeConventions.pdf, and correct the errors.

Needs assessment

Students were grouped in teams of four. Two students played the role of users and two students were the analysts. They were charged with the following task at the beginning of a 75 minute class period. This assignment is loosely based on the XP Planning Game. (7)

You are to develop an ideal student project workroom. The following steps are to be taken.
1) The users should suggest all discrete characteristics of the workroom to the developer analysts. The developers should write down each discrete factor on a 2X2 yellow post-it note. This process should take no more than 10 minutes. The developers should only record during this timeframe. They should not comment or pass judgment.

2) The developers should next present these factors to the users and the users should sort the factors into three categories:
   a. Must have, Nice to have, and Do not need.
   b. The developers should stick these ideas on the wall into each of the three categories as designated by the users. The users should review and finalize this by examining the factors on the wall. This process is given 10 minutes.

3) Now the developers should make an initial attempt at trying to design (draw) a workroom that incorporates all the Must have factors as well as many of the Nice to have factors as possible. This should also take 10 minutes. During design, the users should only observe and not comment.

4) The developers should next present and explain the first design to the users. This process should take no more than 5 minutes.

5) The users then should provide comments and suggestion to the developers. This process should take 5 minutes.

6) The designers should review the suggestions and do redesign for 5 minutes.

7) Steps 4-6 are repeated for another 15 minutes.

8) The third and in this case final workroom should be presented and discussed to the overall class - 5 minutes each.

The total process can be accomplished in approximately 75 minutes.

Function points

A lecture on the concept of function points and an individual exercise in development of a software effort estimation based on factors involved in function points was presented. The requirement was to calculate actual function points as well as person-years required for the project based on given factors. This was primarily an individual exercise but some discussion (collaboration) was allowed. Future courses will include this as a full collaborative exercise.

CONCLUSION

At the conclusion of the course, the students were asked to rate all factors of the course including the collaborative exercises on a scale of 1-5, with 5 most favorable and 1 least favorable. The average ratings for each exercise measured were as follows:

- Coding standards 3.8
- Pair programming 3.7
- Function point analysis 4.0
- Requirements analysis 4.5
- Term project 4.5

This suggests that collaborative exercises were popular and felt to be worthwhile from the students’ standpoint. The overall quality of work performed by the students was judged to be
very good by the instructor. The collaborative experiments were judged overall to be a success and will be included and expanded in future SE course. Future courses will test collaborative methods versus traditional methods to better understand collaborative effectiveness. The author welcomes correspondence with other instructors of SE to share and expand the collaborative possibilities of software engineering education.

REFERENCES