Work In Progress - Motivating Students for Software Engineering

Susan J Lincke
Asst. Prof. of Computer Science, University of Wisconsin Parkside
Kenosha, WI, USA (susan@lincke.org)

Abstract - In basic Computer Science courses, students learn to program small programs with pre-defined programming structures, by themselves. Since software engineering courses teach students to schedule, design, code, and test larger projects in groups, a paradigm shift is required in order for the students to even understand the need for such topics as UML and project management. This paper discusses how to mentally prepare students for such a change, by establishing values, and using reflection, integrity, and process improvement.

Index Terms – Software engineering education, project management, computer science, service-learning.

INTRODUCTION

Students are introduced to Computer Science through programming courses. Programming assignments are single-person projects, usually accomplished in 1-12 hours, with specific directions as to which programming structure(s) to use. After successfully learning to program, they progress to a software engineering class, where we tell them that they must design programs and manage projects. Students wonder: “Why do I need these UML and project management techniques when I have succeeded at programming without them?” While some propose that software engineering techniques should be introduced throughout the curriculum to help eliminate this problem [1-2], Software Engineering instructors are rarely in control of their departments and advanced procedures are rarely needed for simple endeavors. While instructors of other courses simply need to teach relevant material, software engineering instructors must in addition justify to their students why their topic is important. Evidence of this is that Software Engineering instructors know that their most attentive students usually have industry experience...

The attitude problem is only the first of problems to be encountered by a software engineering professor. The computer science curriculum’s reliance on single-person projects causes some students to want to ‘do it themselves’ as opposed to working together. Spoon-fed lectures cause students to be insecure in learning new technical skills on their own. The reliance on small, well-defined problems results in poor designs and no schedule planning. A technical focus results in students despising documentation and, often, arrogant attitudes towards non-technical customers or business managers.

Business leadership training teaches us that leaders must have Vision in order to create an effective culture (or a paradigm shift) [3]. The instructor, as CEO, must instill values into the classroom. Values applicable to a software engineering course include:

- We produce Quality work that fully meets and respects the Customer’s needs
- We achieve our Goals on time and at cost
- We work together as a Team, in a balanced and fair way
- We learn from doing this Project and we learn from our mistakes

These values must be woven throughout the course via exercises, discussion, written reflection, and the grading system.

CREATING THE PARADIGM SHIFT

To create the new culture, the classroom CEO first must answer the unvoiced question: “Show me why I need UML and project management!” This can be achieved by asking the students: “What is the difference between our academic environment and industry?” Through discussion, it arises that projects in industry (pay a salary but) require greater:

- Communication: Teams of people work together on projects
- Complexity: Projects are much more complex and require a longer duration than school projects.
- Quality: All of the product must work, including error and security routines.
- Domain Knowledge: Different application domains make understanding requirements difficult. Customers have diverse goals and skills compared to software engineers.
- Continuous Learning: Legacy software and advancing technologies requires continual learning, often via self-study.
- Predictability: Project funding and success depends upon accurate prediction of project schedules.
- Maintainability: Much time is spent maintaining code, often by persons other than the developer. Therefore the need for accurate documentation is appreciated and helpful in bringing people up-to-speed.

All these requirements lead to an increased need in communication and accuracy than are required for school...
projects. (As a result of classroom discussion, these students raised similar questions to visiting lecturers from industry, in another course. However, the lecturers were applications programmers who worked singly, primarily using component software. Therefore, it is best to emphasize large software development projects in industry.)

The classroom CEO must then show how the course project is similar to what is required of them in industry, and challenge them to try to produce quality work, efficiently and on schedule. Working with real group projects, real customers, and real problems helps to complete the paradigm shift.

Reflection is an important component of authentic learning, because it ties together experience and course concepts [4-5]. Students complete written questionnaires the first and last day of class, which ask open-ended, thought-provoking questions such as:

- What is software quality? How can we achieve software quality?
- What problems are introduced by working in a team environment? How can we minimize these problems?
- How can we deliver software on schedule and shorten project duration?
- How can we best serve our customer?

Reflection can also be discussed in a classroom setting when comparing traditional (e.g., Capability Maturity Model) versus agile (e.g., Extreme Programming) methodologies. How does each methodology achieve quality, communications, schedule accuracy, and customer satisfaction?

Students will not change unless their CEO leader has Integrity [3]. The classroom CEO must practice what they preach in order to achieve consistency, and thereby effective leadership. Any shortcomings of the classroom CEO will result in shortcomings in the classroom (according to [3] and the author’s personal experience...). Inconsistency cannot be disguised or covered up, and the instructor must learn self-discipline and consistency in order to surmount failures in stated goals/values.

Finally, process improvement is an important part of software engineering and can be used to monitor student attitudes. In general, process improvement can be achieved using project deliverables and status reports, customer interviews, instructor evaluations, outside literature, and student questionnaires. Anonymous questionnaires using close-ended questions can quantitatively measure students' attitudes to software engineering in general and to specific practices, both before and after the course is taught. Some statements are obviously incorrect to prevent automatic reactions to answers. Example statements include:

- Schedules are important to keep a project on track.
- Managers should prepare schedules for the people that report to them.
- Engineering should prepare a requirements specification to ensure accurate communication with the customer.
- Design inspections are important in improving quality.
- Adding inspections extends the product delivery time.

Measurement of student attitudes upon entering a course in project management showed that students, on average, ‘Agreed’ to ‘Strongly Agreed’ that formal requirements/design documents, prototypes, and project scheduling were important. However, on average, students had ‘No Opinion’ or ‘Agreed’ that testing, inspections, configuration management, and software metrics were important. At the end of the course, the average score of each of these categories increased to the ‘Agreed’/’Strongly Agreed’ status, except for the testing category – and the importance of prototypes fell to the ‘No Opinion’/’Agreed’ status. This change in attitude very much reflects the type of problems that the students encountered on their projects. On one project, a prototype was not important, but inspections, documentation, and configuration management were paramount. While use of formal configuration management tools and techniques were not a required element of the project, students still learned from the mistakes of the previous project, which handled configuration management poorly.

In one case (of recognizable handwriting), a student’s attitude towards nearly all aspects of software engineering dropped considerably. This student had been a proponent of ‘agile programming’ techniques throughout the course, which for him meant no documentation. The lesson learned is to find an agile software development process that is compatible with classroom projects that involve low design complexity. Such a modification would ensure methodology integrity: students would only need to do what is necessary for the success of their project.

CONCLUSION

Software engineering professors play an important role in shifting the student from small, well-defined projects to larger, more complex group projects. Instructors must decide on a set of values. This paper has discussed how instructors can instill these values into their course and students, via discussion, written questionnaires, the grading system, process improvement, and consistent instructor attitudes.

REFERENCES