Engaging Students in Software Engineering Through Active Learning in Software Engineering

Rajendran Swamidurai and David Umphress

1Mathematics and Computer Science, Alabama State University, Montgomery, AL, USA
2Computer Science and Software Engineering, Auburn University, Auburn, AL, USA

Abstract - This paper presents our three years’ experience in designing and implementing a practitioner-centered based undergraduate software engineering course. To engage the students in software engineering through active learning, we have adapted a single-person software process called Practitioner-Centered Software Engineering (PCSE). The objective of PCSE is to have individuals developers/students enact engineering activities at their own skill level, as opposed to dictating engineering activities in the traditional fashion. As such, PCSE is not a software process, but a framework for identifying an initial software process and modifying that process to meet specific, individual developer/student needs. We describe the implementation of the PCSE-based undergraduate software engineering course, and then present the challenges in successful implementation of PCSE-based pedagogy in undergraduate software engineering courses.

Keywords: Practitioner-centered software engineering; PCSE; active learning; test driven development; software process; personal software process.

1 Introduction

Employment in the software development profession is expected to increase by 22% over the next decade, making it one of the rapidly growing professions in the country. A large increase in the demand for computer software in mobile, healthcare and cyber security industries is the main reason behind this rapid growth in expected employment [1].

Though employment for computing profession is going to increase by 27 percent by the next decade in general, employment of computer programmers is projected to decline 8 percent from 2014 to 2024 [2]. It is clear that U.S. companies don’t want computing graduates who just having programing or coding abilities. Students’ who are inculcated with broader software engineering process technical and business skills than simply programming skills are more attractive to employers [3].

Equipping students with knowledge about practical software development is problematic in today’s academic arena. Colleges teach students the principles of software development, but that instruction is mostly theoretical and abstract. Developing working software requires specific knowledge in software engineering industrial practices. The traditional university curricula do not address these areas in any depth. We have taken a first step in departing from the traditional curricula by orienting an undergraduate course to software engineering practices. Course material on software engineering, including software process, is readily available. What is missing is the mechanism to expose students to real-world software issues encountered in the software industry.

This paper presents an experience in constructing, teaching, and assessing a course that immerses students in real-world software engineering practices through active learning. Our course walked students through producing a working solution by having them use a one-person process, known as Practitioner-Centered Software Engineering (PCSE), developed specifically to apply cutting-edge industry techniques at each point in the software lifecycle. The process was taught incrementally throughout the course, starting with techniques that enhance writing code, and having an engineering activity, which addresses another lifecycle activity added each week. Students worked in an interactive environment in which they were instructed on new techniques and then mentored in the use of those techniques through a series of hands-on exercises.

2 Practitioner-Centered Software Engineering (PCSE)

The Practitioner-Centered Software Engineering (PCSE) is an agile alternative to PSP. It focuses on carrying out fundamental principles of software engineering while letting the developer select appropriate practices. It emphasizes common high-leverage practices by concentrating on high-yield individual performance (which, in turn, can foster team integration) to cultivate discipline, agility, and quality.

PCSE is a tailored collection of different elements from various software processes such as Personal Software Process (PSP) [4], Team Software Process (TSP) [5], Extreme Programming (XP) [6], Feature Driven Development (FDD) [7], SCRUM [8], Rational Unified Process (RUP) [9] etc.
PCSE incorporates the following Capability Maturity Model Integration (CMMI) process areas:

- Project planning.
- Project monitoring and control.
- Risk management.
- Configuration management.
- Process/product quality assurance.
- Measurements.
- Requirements management.
- Technical solution.
- Verification.
- Validation.
- Product integration.

Usage of this model helps the software engineers make accurate plans; consistently meet commitments; improve quality, predictability, productivity, and customer satisfaction; and deliver high-quality products. Using this model, students experience a software development effort from inception to final delivery. It also means that standards and guidelines are small and manageable, and can be distilled for classroom use with little loss of industry intent. This is in sharp contrast to software development using heavy weight software processes in which standards and guidelines are so comprehensive as to be difficult to adapt to student use.

2.1 Software Engineering

Andrew Koenig notes that “software is hard to develop for many reasons: we must figure out what to do, do it, and ensure that we have done it correctly” [10]. The so-called “traditional” engineering fields such as mechanical, civil, electrical, chemical came into being for the very purpose of addressing the same issues. The difference is that the traditional engineering fields address building complex physical things instead of building software. Just as electrical engineering is the business of developing worthwhile solutions through power systems and civil engineering is the business of developing worthwhile solutions through bridges and roads, software engineering can be defined in the same sense: software engineers are in the business of developing worthwhile solutions and they just happen to articulate those solutions in software instead of physical systems.

2.2 Cognitive Activities

The four fundamental activities of engineering are: Envision, identifying a desired future state; Synthesize, determining how to reach that future state; Articulate, taking the actions necessary to reach the future state; and Interpret, ensuring that the future state has been achieved. Applying these activities to software at a high level of abstraction, we have, respectively, analysis, design, construction, and test. This is not suggesting that all these activities take place in sequence. In fact, how we order these activities, how much of each activity we perform and how we carry out each activity determines to a great extent the quality of the software solution itself. Performing these activities in a seat-of-the-pants fashion gives us – and, more importantly our customers – little confidence that we have systematically come up with a good solution.

The term “software process” was coined to describe how and when engineering activities are carried out. The idea was initially exciting due to the parallels it draws to manufacturing processes used in industry to produce quality goods. Over the years, attempts to codify, measure, and enforce process cast a bureaucratic pall over the term, so much so that “software process” is eschewed by many developers because it smacks of attempts to overregulate an essentially creative endeavor.

The tenet engineering activities – envision, synthesize, articulate, and interpret – are abstract in the sense that they address the "what" of problem solving, they don't address how to go about carrying them out. This distinction is important. If an activity is carried out in a defined fashion, we refer to it as a "practice". Put simply: activities are abstract, practices are specific.

The objective of PCSE is to have individuals developers enact engineering activities at their own skill level, as opposed to dictating engineering activities in the traditional fashion. As such, PCSE is not a software process, but a framework for identifying an initial software process and modifying that process to meet specific, individual developer needs. PCSE, thus, views “process” not as prescriptive software development, but as a conscious recognition of the way in which to build software.

For example, suppose we are trying to determine the problem that is to be addressed with a software solution. We are in the "envision" activity, specifically performing requirements analysis. Absent any particular approach to conducting requirements analysis, we consider it an activity with no prescribed practice. If we don't feel comfortable that we know how to accomplish it, we break it down in engineering fashion: we envision, synthesize, articulate, and interpret requirements. In other words, we elicit requirements, analyze requirements, specify requirements, and validate requirements. If we have a specific way of carrying out, say, elicitation, we then say we have a practice for it and we need not define it any further. If we don't have a practice for it, we repeat the engineering process until we get to a level of granularity where we can identify a practice. This process is explained in figure 1.
we have achieved them. The MGI is typically expressed in terms of cost, schedule, and performance.

2) **Minimally Sufficient Activities (MSA)**: The least possible set of conceptual lifecycle activities needed to produce a working solution of a given quality. MSAs describe how objectives will be achieved.

3) **Minimally Viable Process (MVP)**: Orders MSAs so as to structure the effort to be as non-invasive as possible, yet provide enough structure to be viable. Done by listing the lowest-level MSAs and determining work flow. MVP identifies the relationship of the MSAs.

4) **Minimally Effective Practice (MEP)**: Instructs developers in what to do. Done by specifying lowest-level MSAs. MEPs describe how MVPs will be carried out.

The process of identifying the necessary ingredients (MGI + MSA + MVP + MEP) is described in figure 2 and an actual example of PCSE is shown in figure 3.

Having identified the necessary ingredients (MGI + MSA + MVP + MEP) information, we have a basis for alleviating software development pain points. But in an industrial setting, we may not necessarily know the goal process. We start with the status quo and tune according to practitioner needs – hence the name: Practitioner-Centered Software Engineering (Refer figure 4).
4 Implementing PCSE in Software Engineering Course

The Software Engineering II course at Alabama State University is typical in content, focusing on software lifecycle and associated tools and techniques. The instructional language is C++ and the instructional IDE is Eclipse (eclipse.org). Software Engineering II is a 3 credit hour course and meets for 2 clock hours of lecture per week and 2 clock hours of lab per week. All students meet together for the same lecture, but they meet separately in small lab sessions. The course is normally offered every spring semester and each class has no more than 30 students.

For each of the lifecycle topics, students were given a brief description of the relevance of the lifecycle phase and were then walked through an interactive example during class. Concepts were solidified with lab assignments. The
instructional modules and lab materials were prepared by integrating the PCSE practices described above.

The topics presented, in order, were:

- Week 1: Software engineering overview
- Week 2: Lifecycles
- Week 3-4: Eclipse, unittest
- Week 5-7: Construction using TDD
- Week 8: Analysis using scenarios
- Week 9-11: Design using CRC cards
- Week 12-14: Estimation and scheduling
- Week 15: Wrap up and feedback

5 Summary and Conclusion

For the past three years we were using PCSE to teach Software Engineering II. In our first two offering, Spring 2013 and Spring 2014, we introduced the students to the Eclipse platform outfitted with the PyDev Python development plug in. This provided the students an interactive development environment suitable for classwork, and equipped them with skills in working with a software tool in heavy use in industry. The students came to the course with weak development skills and no prior knowledge about Python. This necessitated an unexpected amount of time having to be devoted to using Eclipse and writing rudimentary Python. We had to allow extra time for each assignment, and dropped the last two programming assignments because of lack of time. Since the pre-requisite courses (CSC 211, CSC 212 and CSC 280) for CSC 437 Software Engineering II being taught using C++ language, the student feedback suggested us to use C++ with Eclipse rather than Python with Eclipse. Therefore in year-3, Spring 2015, we offered CSC 437 using Eclipse and C++.

We integrated the PCSE practices discussed above into instructional modules, each lasting one week. The modules were taught using examples that were worked through interactively during class. The students then worked on a programming assignment that incorporated the new instructional concept into concepts previously taught. This allowed the instructors to evaluate the students on their performance and to give points as to how they might improve.

6 Acknowledgment

This material is based upon work supported by the National Science Foundation under Grant No.1238912.

7 References


