2014

An Active Learning Module for an Introduction to Software Engineering Course

A. Frank Ackerman, Ph.D.
Montana Tech of the University of Montana

An Active Learning Module for an *Introduction to Software Engineering* Course

A. Frank Ackerman
Montana Tech

Abstract

Many schools do not begin to introduce college students to software engineering until they have had at least one semester of programming. Since software engineering is a large, complex, and abstract subject it is difficult to construct active learning exercises that build on the students' elementary knowledge of programming and still teach basic software engineering principles. It is also the case that beginning students typically know how to construct small programs, but they have little experience with the techniques necessary to produce reliable and long-term maintainable modules. I have addressed these two concerns by defining a local standard (Montana Tech Method (MTM) Software Development Standard for Small Modules Template) that step-by-step directs students toward the construction of highly reliable small modules using well known, best-practices software engineering techniques. "Small module" is here defined as a coherent development task that can be unit tested, and can be carried out by a single (or a pair of) software engineer(s) in at most a few weeks. The standard describes the process to be used and also provides a template for the top-level documentation. The instructional module's sequence of mini-lectures and exercises associated with the use of this (and other) local standards are used throughout the course, which perforce covers more abstract software engineering material using traditional reading and writing assignments. The sequence of mini-lectures and hands-on assignments (many of which are done in small groups) constitutes an instructional module that can be used in any similar software engineering course.

Overview

The instructional module begins with a small group assignment to look at a small program problem statement and its coded solution, and then to discuss what additional activities/tasks might be applied in the case that this program was a key component in a mission critical system. The instructional sequence continues by using mini-lectures followed by a group or individual activity that addresses the particular software engineering technique covered in the mini-lecture. The example used in the mini-lectures is simpler than the problem the students are addressing. Software inspections are introduced early in the module and are used across groups as a technique for focusing critically on the work of each group. In this way the module covers requirements (at the small module level); design (using simple UML diagrams and a standardized pseudo-code language); unit test design (constrained by requiring a complete, logical, hierarchical partitioning of the input space); test cases for each leaf partition; coding to a specified standard; coverage testing (including random test case generation); and correctness proofs. Along the way the concept of size and effort estimation is introduced as well as the necessity of capturing in-process data for estimates-to-actuals comparison.

The final exercise in this module is an individual (or pair) assignment to take another problem with a small module solution and apply the module requirements, design, and test case design and execution techniques covered in the module (time constraints have

978-1-4799-4970-0/14 © 2014 IEEE CSEE&T 2014, Klagenfurt, Austria

so far prohibited the use of proofs of correctness and inspections in this exercise). Finally, a class period is devoted to one-by-one submitting the completed programs to a robot judge for either an "accepted" or "reject" verdict and keeping score – all of the excitement of a TV quiz show!

The MTM standard provides a framework for all the commonly known module level best practices techniques which I have found over many decades of experience to be effective in the development of highly reliable software modules. The keyword is "framework." The standard, and specifics of each technique, can be modified to suit a wide variety of software engineering teaching or development environments. All of the materials used or referenced in the presentation are available from the Montana Tech Computer Science Department web site under a Creative Commons Attribution-ShareAlike License, and thus can be modified as desired for particular teaching or development environments.

Please remember that these materials are for novice software engineers. My experience is that providing a very specific documentation framework for this level of student enables them to concentrate on effectively learning the techniques of software engineering that are covered in the instructional module.

How effective is the set of techniques covered in this presentation? How does one measure effectiveness? The effectiveness measure that I use is that upon initial delivery (and the delivery of any subsequent modifications), the software will fully satisfy all run-time functional requirements (the verification of all other requirements can be addressed in reviews). In my case, I use the UVa (`uva.onlinejudge.org`) competitive programming site to assess the initial delivery acceptance rate as described above. I just introduced the use of the UVa repository the last time I taught our introduction to software engineering course, so I have just one hard data point: my students had better than a 70% "first shot" acceptance rate. The overall UVa acceptance rate for this problem (by competitive programmers all over the world making multiple submissions was, at the time a little better than 45%. The next time I teach our introductory software engineering course I will use a standardized programming aptitude test at the start of the course, and will begin this module by having the students attempt a judged solution before working on the techniques covered in this instructional module. It would also be most helpful if other faculty teaching similar courses would use their own modifications of this instructional module and collect and report capability and effectiveness data on their classes.

The techniques covered in this presentation are aimed at achieving the highest possible reliability, not the shortest development time. However, in cases where reliability may be traded off for quicker delivery some techniques (*e.g.*, correctness proofs) may be dropped and some of the techniques may be modified (*e.g.*, unit testing) to reduce development time. However, the basic principles of software engineering should be adhered to. That is, these techniques are not applicable to one-shot, hack-out-as-quickly-as possible software.

This presentation will be accompanied by a fully completed MTM standard template example, and standardized code and unit tests for a small module. The presentation will explain each section of the completed template; whether or not it is an essential component of my set of techniques, and how a section might be modified for a variety of teaching or development environments.