Towards Teaching Software Craftsmanship

Todd Sedano

February 22, 2012

CMU-SV-12-003

Carnegie Mellon University
Silicon Valley

Silicon Valley Campus
Carnegie Mellon University
Mountain View, CA 94035

Towards Teaching Software Craftsmanship

Todd Sedano
Carnegie Mellon University
Silicon Valley Campus
MS 23-11
Moffett Field, CA 94035
011-650-335-2812
todd.sedano@sv.cmu.edu

Abstract

We report on the experience of the first offering of the course, "The Craft of Software Development." The purpose of the course is to identify and remediate individual weaknesses as software developers.

Each student was asked to pick a personal learning focus in an area of known software development weakness for that student. Through a "maiden speech" they asked their class community for help in creating a learning plan to address their area of weakness. Learning plans typically incorporated known apprenticeship patterns such as "Find a Coach," "Breakable Toys," and "Practice, Practice, Practice." After creating their learning plan, students identified metrics to track their progress towards their goal. While executing their individual learning plans, the class performed programming katas which are specific programming exercises to work on issues relevant for the entire class.

Several issues emerged in the first offering in the course and several suggestions are provided for incorporation in the next offering of the course.

1. Introduction

Students pursuing software engineering as a profession resonate with the need to improve their own craft. It has been conjectured that in order to become an expert, practitioners need to spend roughly 10,000 hours or 10 years of deliberate practice and coaching. [1] While incorporating practice into one's routine is good, "not all practice makes perfect." People tend to practice activities at which they are proficient, which is more enjoyable than working on weaknesses. In contrast, in deliberate practice, a coach guides a practitioner through specific activities for a task that the practitioner wants to improve or can't even do today.

Early in the journey, a coach gives the practitioner specific exercises that address known weaknesses. Eventually the practitioner will outgrow the expertise of the coach and will need to find a new coach. When the practitioner nears mastery and there are no more advanced coaches, then the role of the coach is to teach the practitioner how to be their own coach. For the craft of software development, the ideal coach has progressed through their own journeyman adventure, can identify areas of improvement, and can suggest appropriate activities for the practitioner to do. One path of deliberate practice is for the practitioner to find a coach, identify an area to improve upon, create a plan, implement the plan, collect metrics for assessment, and reflect upon the experience.

2. Course Overview

At Carnegie Mellon University in Silicon Valley, we offered a seven-week experimental course called "The Craft of Software Development" to 11 of our masters students during

Spring 2011. Since these students are enrolled in our MS Software Engineering - Technical Track [2], they want to be technical leaders who love writing code and want to influence the technical direction of their workplaces. We created “The Craft of Software Development” to identify ways to enhance their programming skills and practice patterns that will further their careers.

The goals of the course were to:

- identify a learning focus (an area of weakness) and create a learning plan to achieve that goal using known apprenticeship patterns
- determine metrics to see if they are achieving their goal
- find a coach or mentor
- incorporate deliberate practice into their routine
- improve self-directed learning skills

3. Course Mechanics

3.1. Identify a Learning focus

At the start of the course, each student selected a learning focus, an area of known weakness. Examples of skill-based learning foci include “Practice Test-Driven Development”, “Write readable code”, “Write DRY code” (Don’t Repeat Yourself). Examples of technology-based learning foci include “Learn C++”, “Learn Ruby”, “Learn Python”, “Become a Ruby on Rails expert”, “Develop Android apps”, “Learn Objective C and iPhone development”, and “Create custom UI components in Apple’s iOS.” Students could work in any programming language.

3.2. Create a Maiden Speech

During the first week of the class, each student created a "maiden speech" that announced to the class the student’s learning focus. (A “maiden speech” is the first speech an elected politician makes discussing their platform and items they want to accomplish during their term in office.) Students crafted their maiden speech in the form "For me software craftsmanship includes the notion of _____. I want to get better at _____. What does the community think will help me improve_____." [3] For example, one student said: “For me software craftsmanship includes the notion of 'test-driven-development' as a means to help me improve my testing skills. Often I find myself writing code in order to get it done and to my users quickly; I'll do manual testing, but inevitably my users find bugs that could have been detected earlier. I want to get better at 'test-driven-development'. What does the community think will help me improve in this area?” For some developers, asking one’s community for help is difficult and a significant step.

3.3. Create a Learning Plan

In order to achieve their goal, students identified specific steps and created a learning plan with feedback from the student community. Students were encouraged to incorporate apprenticeship patterns. [4] See Table 1 for common patterns chosen by the students.

Table 1. Apprenticeship Patterns

Pattern Name	Purpose
Find a Coach	Often we know where we want to go, but not how to get there. An experienced coach suggests activities of deliberate practice.
Practice, Practice, Practice	Often when we do something once, we notice the novelty. By repeating the same activity over and over again, we focus on ways to improve our workflow.
Expose your ignorance	By being willing to talk about things we are weak in, we're able to accept help and advice from others.
Share as you learn	By recording and sharing what we learn, we create a record of how things went which can be helpful for ourselves and others who follow the same path.
Kindred Spirits	By hanging out with people who have the same problems, we're exposed to new ways to improve our own craft.
Breakable Toys	Before we use something for real, it's good to practice with it on a project that doesn't really matter.

The course required all students to "Find a Coach."

3.5. Identify Metrics to Track Progress

Each student created metrics to track progress towards achieving their goal. At the end of the course, the metrics evaluated how much improvement was made. The students struggled with this even though they completed a metrics for software engineers course. The metrics were not Specific, Measurable, Actionable, Realistic, and Timely (SMART) metrics. [5] After a class discussion about SMART metrics, we provided the following metrics as a starting point for the most popular learning objectives. The students then reworked their metrics.

Test Driven Development: How many methods were tested before the code was written?

Learning Ruby: How many times did I look up APIs?

Paired programming: How many of the methods are written using paired programming?

Readability: How many times do other programmers say that my code is readable?

Don't Repeat Yourself: How many instances are there of copy-and-pasting of information?

3.6. Perform Katas

To provide structure and a rhythm to the course, we did code katas, programming exercises designed to improve skill through repetition [6], suggested on the Software Craftsmanship google group mailing list. [7] For example, the Gilded Rose kata [8] illustrates the problem of encountering production code that contains complex nested if statements. We discussed the experience of each kata in class. Many students had similar experiences with a particular kata.

For example, Gilded Rose showed the power of writing test cases before making modification to existing code in order to improve confidence that the changes worked.

4. **Issues and Next Steps**

In the first offering of the Craft of Software Development, we encountered several issues that we will address with the next course offering.

Problem: Students selected multiple learning foci, which distracted their attention.

Next time: Require that students pick a single learning focus.

Problem: Several students selected a learning focus related to a specific technology, (e.g., "Learn Ruby") which is not an area of weakness, per se, for most software developers. Students who picked a resume enhancement opportunity missed a chance to practice and improve their craft. In retrospect, this dynamic isn't a surprise since novices have poor metacognitive skills in recognizing in personal weaknesses. [9] In using deliberate practice, a coach, not the student, would identify areas of weakness and help develop a series of practices to address short comings.

Next time: We're planning to augment the Code Retreat format [10] where developers follow pair programming through six coding iterations in one day. We'll add a step where each programmer describes one thing their partner did well and one thing to improve. Students will then be encouraged to work on a peer-identified area for improvement.

Problem: Students produced vague learning plans that required rework (e.g., "I'm going to do pair programming" which does not provide guidance on what kind of code to write, or how to find programming partners.) Perhaps a vague understanding of their learning focus manifested itself in vague plans. Some students confused their objective and recorded it as their plan. (e.g., "[Be] able to write code which is readable.")

Next time: Either provide sample learning plans or have students swap learning plans and provide feedback on whether they could execute someone else's learning plan as their own.

Problem: Most students struggled with the process of finding a coach. In class, we discussed strategies and the benefits of this career skill. More than half of the students did not find a coach. One felt that interacting with the faculty would count as finding a coach. Another felt that as a beginner he didn't need a coach. These explanations illustrate that the course did not experientially reinforce the need for deliberate practice. At a cognitive level, most understood the importance of a coach, but this didn't translate into a deep seated conviction that resulted in action.

Next time: Finding a coach is challenging for a working professional, let a lone a graduate student taking a course on a tight timeline. Expecting the student to accomplish this in the first two weeks is unrealistic. If the student knows an area of weakness prior to the course, finding a coach immediately for deliberate practice may be less important. Next time, we'll ask students to contact alumni as possible coaches.

Problem: As discussed in section 3.5, most students struggled with creating metrics.

Next time: Provide sample metrics and reinforce Goal, Question, Metric (GQM) [11]

Problem: Not all the katas were useful. We did the Prime Factors, Gilded Rose, Potter, and Across the Board [12] katas. All worked really well except for the Across the Board kata for which we spent two class sessions.

Next time: we'll replace Across the Board with two different katas.

5. **Conclusion**

The process of identifying an area of weakness, creating a plan, and tracking metrics is useful for any software developer on their journey to become software expert. The Craft of Software Development provided the chance to apply learning patterns that students can use for the rest of their careers while improving on an immediate area of weakness, and perhaps more importantly, the course trained the students with practical skills in self directed learning.

This first course offering achieved many of the course learning objectives. Most students did identify a learning focus and executed a plan based on apprenticeship patterns. Most students struggled with creating metrics and finding a coach. A few incorporated deliberative practice into their routine. Many improved their self-directed learning skills.

The students enjoyed taking this course. 83% of the students (response size of six) would recommend this course to a peer. The one student who would not recommend it felt that there should be more structure in the course. Those who did recommend it perceived it extremely relevant to their careers and enjoyed the flexible, yet structured framework for students working on diverse learning goals. We plan to offer the course again in Spring 2012. We are considering making this a required course. Finally, we enjoyed watching our students improve their software development skills and enjoyed teaching this course.

6. **References**

[1] K Ericsson, M. Rietula, and E. Cokely, "Making of an Expert" Harvard Business Review. August 2007.

[2] R. Bareiss and T. Sedano, "Developing Software Engineering Leaders," Proceedings of the First International Symposium on Tangible Software Engineering Education (STANS-09 in Tokyo,Japan), October 2009. (http://repository.cmu.edu/silicon_valley/27/)

[3] A. Green, "An indecent proposal" https://groups.google.com/group/software_craftsmanship 1/15/2011.

[4] D. Hoover and A. Oshineye. Apprenticeship Patterns. O'Reilly. Cambridge, MA. October 2009.

[5] G. Doran. "There's a S.M.A.R.T. way to write management goals and objectives." Management Review. November 1981.

[6] software_craftsmanship@googlegroups.com

[7] D. Thomas, "Code Kata," January 2007. (http://codekata.pragprog.com/codekata/)

[9] T. Hughes and B. Johnson, "Gilded Rose Kata," February 2011. (http://craftsmanship.sv.cmu.edu/posts/gilded-rose-kata)

[9] J. Kruger and D. Dunning. "Unskilled and Unaware of It: How Difficulties in Recognizing One's Own Incompetence Lead to Inflated Self-Assessments". Journal of Personality and Social Psychology. 1999.

[10] http://coderetreat.com

[11] V. Basilli, G. Caldiera, H. D. Rombach. "The Goal Question Metric Approach." Encyclopedia of Software Engineering, pp. 528-532, John Wiley & Sons, Inc., 1994.

[12] http://craftsmanship.sv.cmu.edu