Expert Code Review and Mastery Learning in a Software Development Course

Sophie Engle
sjenngle@cs.usfca.edu

Sami Rollins
srollins@cs.usfca.edu
INTRODUCTION

Why Try Mastery Learning?
Motivation

• Some students made it to upper-division courses, but unable to pass those courses on first try
  – Most delay graduation and retake
  – Many upper-division courses offered once a year
• Often have weak but broad level of programming
• Somewhere between start and end of degree, not quite preparing these students for harder classes
Software Development

• Bridges lower division programming courses and upper division programming-heavy courses
• Provides student **project experience**
• Goal is to produce **well-designed** large software project, approximately 2k lines of code
• Promotes **iterative development**
• Undergraduates already have two introduction to programming courses (Python and Java)
Issues Identified

• Possible to gain enough partial credit to pass the course without mastering all of the core concepts
• Easy to test for correctness, difficult to test design
  – Unit tests and scripts for correctness
  – Code review for design
• Assigning partial credit to code design tricky
  – Teacher assistants unfamiliar with code review
  – Unwilling to give low grades for functional code
Issues Identified

• Originally believed iterative projects would lead to iterative development
  – Students loath to refactor "working" code
  – Students not sure of issues and how to fix them

• Only a certain core of projects were really critical for upper division courses
  – Needed multithreading, code design mastery
  – Did not need mastery of web-related topics
Hypothesis

• Use homework and quizzes to address breath, and projects for depth

• Use **mastery learning** to force students to master core concepts necessary for upper division courses

• Use **expert code review** to enforce mastery learning for code design

• Force students to **refactor code** until passes both unit tests and code review
BACKGROUND

Software Development Course
Course Basics

• Semester-long course
• Hour-long classes meeting three times a week
• Approximately 10 to 30 students per section
• Offered every semester
• Mostly undergraduate majors (part of core)
• Also included minors and new graduate students*

*We revisit this later on in the talk.
Environment

http://cs.usfca.edu/facilities.html
Traditional Approach

• Lecture
  – Hour long twice weekly
  – Slide-based or live code walkthroughs

• Lab
  – Hour long once weekly
  – In-class homework and quizzes*

• Exams
  – One midterm and one final exam
  – Closed-book closed-note except Java API

*These were added in later versions of the course.
Traditional Approach

• Seven large iterative programming projects
  – Word Count
  – Inverted Index
  – Partial Search*
  – Multithreading
  – HTML Parsing
  – Web Crawler
  – Search Engine

• Assigned throughout semester

*Used to include a redesign component.
Student Experience

• Very popular among students
  – Helpful in future courses
  – Helpful for finding software development jobs

• Very motivated by search engine project

• Reputation for being fairly easy to pass
  – Kludge together something before deadline
  – Get partial credit and move on to next project
APPROACH

Mastery Learning and Expert Code Review
Approach

• Traditional Approach (Breadth)
  – Lectures
  – Homework
  – Quizzes
  – Exams

• Mastery Learning (Depth)
  – Projects
Projects

• Reduced from 7 to 5 projects
  – Keep inverted index, partial search, multi-threading, web crawler, and search engine
  – Assign word count and HTML parsing projects as homework instead

• Two-stage project submission
  – Teacher assistant runs tests for correctness
  – Instructor performs code review for design

• Must continue to refactor until both stages pass
Projects

• Unable to start next project until current passes

• Provide *suggested deadlines* to try and keep students on track

• Accelerated deadline schedule to promote an agile approach and provide time for resubmission
  – Students instructed to expect to submit twice
  – Students must also master time management

• Cutoff deadlines given to ensure enough time for resubmission and still pass
Project Grading

• Projects are worth majority of grade

• Project grade based on how many projects passed
  – Must pass multithreading project to pass course
  – If perform poorly on exams, must also pass web crawler to pass the course

• Small penalty deducted if students resubmit project too many times*

• Small extra credit added if students submit by suggested deadlines*

*Again, this reflects how it is done currently. It was slightly different in previous semesters.
Code Review

• Performed by instructor, not teacher assistant
  – Instructor has code review experience
  – Instructor more strict on design and style

• Performed interactively with student*
  – Each session maximum 20 minutes
  – Specific criteria evaluated for each project
  – Comments made directly in students' code and committed to their svn repositories

• Result is either pass, conditional pass, or resubmit

*This is how we are doing it this semester, which has evolved since when we wrote the paper.
Review Criteria

• Assume once criteria passed in one project, will be correct in following projects
  – Not ideal, but necessary due to time constraints

• Inverted index (first project) criteria
  – Proper code style (e.g. comments, names)
  – Proper use of keywords (e.g. static)
  – Proper generalization (e.g. reusable code)
  – Proper encapsulation (e.g. no passing references of private mutable members)
RESULTS
Grades, Submissions, Lessons Learned
Measurements

• Compared two semesters
  – Fall 2011 using traditional approach
  – Spring 2012 using expert code review
• Evaluated student performance
  – Number of submissions
  – Average project grades
  – SLOC per project
• Evaluated student experience
  – Conducted survey
Caveats

• Difficult to get statistically significant results!
  – Small classes sizes to begin with (≤ 30)
  – Did not include graduate students
  – Did not include minors
  – Did not include ghost students*

• Only 9 students for Fall and 12 for Spring semester

• Different types of students in Fall versus Spring
  – Fall had separate section for graduates
  – Spring combined undergrads and grads

*Some students never showed up, or dropped before the first project suggested deadline.
Results

Provided for Spring semester (with mastery learning and expert code review) only.
## Results

<table>
<thead>
<tr>
<th>Project</th>
<th>#S</th>
<th>SLOC</th>
<th>Grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>1: Inverted Index</td>
<td>3.1</td>
<td>186</td>
<td>218</td>
</tr>
<tr>
<td>2: Partial Search</td>
<td>2.8</td>
<td>341</td>
<td>418</td>
</tr>
<tr>
<td>3: Multithreading</td>
<td>2.0</td>
<td>494</td>
<td>706</td>
</tr>
<tr>
<td>4: Web Crawler</td>
<td>1.2</td>
<td>686</td>
<td>914</td>
</tr>
<tr>
<td>5: Search Engine</td>
<td>1.0</td>
<td>2360</td>
<td>1781</td>
</tr>
<tr>
<td><strong>Average:</strong></td>
<td>2.5</td>
<td>569</td>
<td>590</td>
</tr>
</tbody>
</table>

See discussion in paper.
Comments

• "I had fun with the projects and they made me work hard."

• "Projects were more challenging because there would usually be significant refactoring that had be done after each grading session."

• "The hardest thing was the grading process of the projects, it takes way too long for the resubmission process to take place."*

• "I really understand the idea of object oriented programming after CS212."

*A very common complaint.*
Conclusions

• Mastery of code design and refactoring improved
  – Supported by decreasing number of submissions

• Mastery of complex concepts improved
  – Supported by higher grades of first three projects

• Project progression was slower
  – Supported by lower grades of last two projects

• Time required for the code review was reasonable
  – Maximum 30 students, 20 minutes per review
  – Only subset of students need review each week
Drawbacks

• Time Issues
  – Time consuming to setup and get process down
  – Must coordinate verification and code reviews to avoid major delays
  – Students must wait longer for grades

• Attrition
  – Better prepares majors, what about minors?
  – Students still fail due to poor code, but now also poor time management
Drawbacks

• Progression
  – Difficult to reset student expectations
  – Difficult to acclimate students to new process
  – Disbelief that I will force them to resubmit when they already have "working" code
  – Difficult to convince students they are running out of time for submissions

• Evaluation
  – Difficult to calculate student grade mid-semester
CONCLUSION

Summary and Final Thoughts
Summary

• Some students failing upper division courses
• Focus on bridge course between lower and upper division courses
• Keep traditional approach for breadth on lectures, homework, quizzes, and exams
• Use mastery learning enforced via expert code review for projects and code design
• Some initial success of approach
Adaptation

- Requires low faculty to student ratio
- Requires space in schedule for resubmission
  - Difficult on quarter schedules
- Requires incremental projects
  - Otherwise difficult to justify refactoring
- Requires appropriate subset of topics for mastery
- Requires mid-level course
  - Expert code review less necessary for lower levels
Future Directions

• Before Class
  – Watch slide-based lectures
  – Watch short code walkthroughs

• Class Time
  – Lab exercises and quizzes
  – Longer code walkthroughs

• Code Reviews
  – Every other week despite project status
    (30 students, 20 minutes, 5 hours weekly)
MOOC Comparison

• Low enough faculty/student ratio for more one-on-one interaction than possible with MOOCs

• Better assessment of student status
  – Automatic assessment of homework and quizzes
  – Manual assessment of projects and exams

• Still get benefit of recorded videos
  – Students can easily re-watch videos
  – Frees up class time for other activities
Questions?

• **Sophie Engle**  
  sjengle@cs.usfca.edu  
  http://sjengle.cs.usfca.edu/

• **Sami Rollins**  
  srollins@cs.usfca.edu  
  https://sites.google.com/site/srollins/

• **Course Website**  
  http://cs212.cs.usfca.edu/