Crowdsourced Code Review in Programming Classes

Rob Miller
MIT CSAIL

joint work with Mason Tang, Elena Tatarchenko, Max Goldman, Joe Henke
Problem: Feedback about Coding Style

• MIT 6.005 Software Construction
  – foundation-level programming course (replaced 6.001/6.170)
  – 400 students per year, mostly sophomores
• Students write lots of code
  – roughly 10kloc in problem sets and projects
• Automatic grading is necessary but not sufficient

```c
// compute n! requires n >= 0
int factorial(int n) {
    if (n == 0) return 1;
    else return n * factorial(n-1);
}
```

```c
int factorial(int n) {
    int i, result=1;
    if (n == 0) result = 1;
    else {
        for (i = 1; i < n; ++i) result *= i;
        result = result*n;
    }
    return result;
}
```

– we need human readers, and we want line-by-line feedback
Approach: Crowd-Driven Code Review

- Chop up student programs into **chunks**
- Review the chunks by a **mixed crowd**: students, staff, alums

- Anticipated benefits
  - faster, cheaper, more diverse comments
  - give practice with code reviewing (a widespread industry practice)
  - expose to good and bad solutions
  - reduce workload on teaching staff
  - incorporate alumni back into the course

- Not using for grading... yet
Outline

✓ Introduction
  • System
    – Caesar code-reviewing system
  • Process
    – How we use Caesar in a programming class
  • Experience
    – Some results from initial deployment last fall & spring
  • Looking ahead
    – Issues to address
    – Opportunities for on-campus and online
Caesar: Divide & Conquer

```java
public class RulesOf6005 {

    /**
     * Tests if the string is one of the items in the Course Elements section.
     */
    @param name - the element to be tested
    @return true if <name> appears in bold in Course Elements section. Ignores case (capitalization).
    * Example: 'Lectures' and 'lectures' will both return true.
    */
    public static boolean hasFeature(String name) {
        String[] elements = { "lectures", "recitations", "laptops required", "text", "problem sets", "";
        for (int i = 0; i < 9; i++) {
            if (elements[i].equals(test)) {
                return true;
            }
        }
        return false;
    }

    /**
     * Takes the quiz, pset, project, and participation grades as values out of a hundred.
     * Behavior is unspecified if the values are out of range.
     */
    @param quiz
    @param pset
    @param project
    @param participation
    * @return the resulting grade out of a hundred.
    */
    public static int computeGrade(int quiz, int pset, int project, int participation) {
        return (int)Math.round((quiz*.2) + (pset*.4) + (project*.3) + (participation*.1));
    }

    /**
     * Based on the slack day policy, returns a date of when the assignment would be due, making sure not
     * exceed the budget. In the case of the request being more than what's allowed, the latest possible
     * due date is returned.
     */
    * Hint: Take a look at http://download.oracle.com/javase/6/docs/api/java/util/GregorianCalendar.html
    * Behavior is unspecified if request is negative or due date is null.
    */
```
Social Reviewing

reviewers can see whole program (not just chunk) if needed

reviewers have a reputation (#upvotes, + 100 if they’re alums or staff of the course)

upvotes & downvotes

downvotes

replies & discussion

replies

reviewer comments

reviewer comments

upvotes

automatic style checker comments
Code Chopping

- Chop each submission into chunks
  - We’ve tried two sizes: method (~20 loc) and class (~150 loc)
- Sort chunks
  - Prioritize classes marked important by staff
  - Diff against staff-provided code and prioritize chunks with more student-authored lines
  - Deprioritize test code
  - Cluster chunks using MOSS fingerprinting
- Not all chunks get assigned for review
  - Typically 30%
  - So prioritization is important
Task Routing

• Lazy assignment
  – wait until a reviewer logs in before assigning tasks to them, since a given alum/student may or may not show up

• Multiple reviewers per chunk
  – assign 2 students/alums to a chunk to encourage discussion
  – assign 1 staff to the chunk too, for reviewing the reviews

• Fairness to code author
  – balance #reviewed chunks per submission

• Diversity of learning opportunities
  – maximize differences between reviewers on same chunk & same submission
    • role (student, alum, staff)
    • reputation score
    • # times assigned to a chunk together
Privacy vs. Visibility

- Code author is anonymous
- All reviewers are identified & their reviews are browsable

**Robert C Miller**

**rcm**

reputation: 107

<table>
<thead>
<tr>
<th>ps3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Also, it might be a good idea to append some sort of general error indicator to the except...</td>
</tr>
<tr>
<td>I would suggest against catching every error of every possible type; instead, focus on cal...</td>
</tr>
<tr>
<td>What if you get a blank string, or a string with improper grammar?</td>
</tr>
<tr>
<td>Looks good. Simple and elegant. Are you throwing only RuntimeExceptions? I'm assuming your...</td>
</tr>
<tr>
<td>Why not just throw the exception? What if there's an error in the lexer?</td>
</tr>
<tr>
<td>Your specs could use a little more detail, but the code looks fine.</td>
</tr>
<tr>
<td>Right. Rep exposure!</td>
</tr>
<tr>
<td>Looking at your implementation of Parser, this would be okay if result were final. However...</td>
</tr>
<tr>
<td>The fact that this can return an error should be in the spec.</td>
</tr>
<tr>
<td>I would argue that this exception should be thrown rather than caught. Otherwise, anyone...</td>
</tr>
<tr>
<td>but would be better to throw as an exception rather than returning as a string. Likewise...</td>
</tr>
<tr>
<td>That's fair. Hmm, maybe the course instructors should not put code in Caesar if the author...</td>
</tr>
</tbody>
</table>
Process

• Assignment due on Thurs

• Students review Fri/Sat
  – 10 chunks; must comment or vote on each
  – assignment is still fresh in their minds, curious about other’s solutions

• Staff reviews Sun
  – 30 chunks; mostly voting

• Alums review whenever (Fri-Wed)
  – 3-5 chunks; must comment or vote

• Revise & resubmit assignment by Thurs
  – Main motivation is to fix bugs that lost points in autograding, but must address code review comments as well
Experience

180 students
54 alums
15 staff

Fall 2011

215 students
0 alums
17 staff

Spring 2012

13 problem sets, 2.2k submissions

21.5k comments
5% alums
8% staff
87% students

16.2% upvoted
0.7% downvoted

9.6 comments per submission
Kinds of Comments

Bug
Clarity
Performance
Simplicity
Style
Learning
Positive
~13%
LGTM
< 1%
Meta

• nice use of comments
• I like how you have assert statements sprinkled throughout your test: Great way of checking where things are failing.
• looks good to me
• Test looks fine
• Two of my files to review are Main :[
  • I don't think I should be reviewing this method. I think its a problem with caesar.
Interaction Happened

- 16% of comments were upvoted
- 22% of comments were in conversations (threads with at least 2 human comments)

- Corrections
- Clarifications
- Author responses

- **student**: Prefer equalsIgnoreCase because it says to ignore capitalization
  - **student**: The name = name.toUpperCase line seems to take care of capitalization.
- **student**: Agreed. I think the desired method here is add, not roll
- **student**: what does this method do?
  - **code author**: Essentially assertEquals. As someone else pointed out, this should have been an assert array equals, so that will happen in the future.
Alums Have Different Context

- **alum**: avoid abbreviating variable names... 'hi' would be especially confusing to someone who isn't familiar with English.
- **student**: I dk about this one though. I've seen lo and hi at various places. I'd say this is fine. Though, the integer N should be lower case, just to conform with the java naming convention.
- **alum**: When you're in industry and working on code with people who aren't necessarily familiar with English... it's a problem to use phonetic abbreviations. There is really no good reason to abbreviate variable names especially since IDEs autocomplete.
- **student**: For that, yes, I agree. Sadly, some IDEs don't autocomplete very well. *cough*
- **code author**: These were the variables that were given to us.
• **staff:** A magic number is a some fixed constant (numbers, Strings, etc.) used in a place where variables would likely add to code clarity or understanding...

• **staff:** Magic numbers are simply primitives that are used without having a descriptive variable name...

• **alum:** Returning null is usually dangerous, especially if your Javadoc doesn't mention that it might return null in some cases...

• **alum:** Don't return null in exception cases, especially if your Javadoc's @return line doesn't mention the possibility of returning null (even though it's mentioned earlier)....

• **student:** Should include more in rep invariant. ie a digit is allowed only once per row, column, and box

• **student:** Should include more in rep invariant. ie a digit is allowed only once per row, column, and box
Some Code Authors Are Self-Conscious
Survey Responses

• Learning
  – I thought it was helpful. There are **more things I'll be looking for when I code because I saw them in other people's code and people commented** about it.
  – I like how reviewing the code after we've submitted it **allows us to retain the information** and concepts for a longer time.

• Self-doubt
  – I **think I have more to learn than to offer**, at this point though.
  – I was glad that the **code I was assigned were on the parts that I had understood** in my submission, so I could give better feedback.
Survey Responses

• Skepticism
  – Meh. I don't think this will have a major impact on students' coding.
  – No; I'm fixing code instead of learning.

• Keen to give and get more
  – Is it alright to review more than what is assigned on Caesar? I really like reading and reviewing other people's code, but don't know if over-reviewing would be frowned on.
  – I'm thrilled to get 51 human comments, but they are ALL, somehow, on tests.

• Scary
  – Very good UI. I enjoy commenting on code. I feel like the Simon Cowell of code review.
Related Work

• ColorMyGraph [Blank, Sutner, von Ahn]
  – crowd grading of proofs

• Github
  – social commenting on code commits

• Code Review StackExchange
  – threaded Q&A about short code chunks
Looking Ahead

• System changes
  – Capturing common advice in a wiki
  – Increasing reviewer leverage: searching for similar code
  – Promoting good comments for all users to read
  – Alumni engagement: recruiting, social networking, reputation management

• Other possibilities
  – Combining automatic grading (e.g. online tutors) with crowd-driven human feedback

• System is open source
  – http://github.com/uid/caesar-web