Teaching Composition Quality at Scale

Human Judgment in the Age of Autograders

John DeNero
Computer Science Division
University of California, Berkeley
Berkeley, CA
denero@cs.berkeley.edu

Stephen Martinis
Computer Science Division
University of California, Berkeley
Berkeley, CA
stephenmartinis@gmail.com

ABSTRACT

We describe an effort to improve the *composition quality* of student programs: the property that a program can be understood effectively by another person. As a semester-long component of UC Berkeley's first course for majors, CS 61A, we gave students composition guidelines, scores, and qualitative feedback—all generated manually by a course staff of 10 graders for over 700 students.

To facilitate this effort, we created a new online tool that allows instructors to provide feedback efficiently at scale. Our system differs from recently developed alternatives in that it is a branch of an industrial tool originally developed for internal code reviews at Google and used extensively by the open-source community. We found that many of the features designed for industrial applications are well-suited for instructional use as well. We extended the system with permissions controls and comment memories tailored for giving educational feedback.

Using this tool improved the consistency of the feedback we gave to students, the efficiency of generating that feedback, and our ability to communicate that feedback to students. Emphasizing composition throughout the course improved the composition of our students' code. The quality of student programs improved by a statistically significant margin ($p < 0.01$) over those from a previous semester, measured by a blind comparison of student submissions.

Categories and Subject Descriptors

K.3.2 [**Computer Science Education**]

General Terms

Human Factors

Keywords

Introductory Programming, Web-Based Feedback

SIGCSE'14, March 5–8, 2014, Atlanta, GA, USA.
Copyright is held by the owner/author(s). Publication rights licensed to ACM.
ACM 978-1-4503-2605-6/14/03 $15.00.
http://dx.doi.org/10.1145/2538862.2538976.

1. INTRODUCTION

For the programs used most widely in the world, their source code is read by many developers. By contrast, student assignments are typically read at most once by an instructor. In large courses, evaluations of correctness are performed most efficiently by scripts, rather than instructors, as the effort of autograder development is amortized across a large number of students [4]. Although autograding has tremendous benefits [11, 7, 6], it can potentially deemphasize the importance of writing programs that other people can understand. This paper describes a software tool and associated policies that mitigate this effect by explicitly promoting *composition quality*—that entirely subjective property of a program that it can be understood easily by a person.

Efforts to encourage students to write comprehensible programs can be traced back through decades of computer science education (e.g., [1]). Teaching composition quality necessarily involves manual feedback—a fellow human is the best judge of whether a program can be understood by people. In CS 61A, UC Berkeley's first course for majors, we have always provided some manual feedback. In Fall 2012, we improved this component of the course by developing a software tool that manages the feedback work flow in tandem with autograding scripts that perform correctness evaluation. The tool was designed to scale to large courses. In CS 61A, more than 3000 project submissions from more than 700 students were evaluated by 10 student graders.

Our web-based tool is designed to promote consistent evaluation, an efficient process, and smooth communication with students. It is a branch of an existing industrial tool and therefore includes many features originally designed for a professional setting, such as integration with multiple version control systems. We extended the open-source Rietveld[1] project, which was originally developed for code reviews at Google and is used widely in the open-source developer community. We found many features of this system to be useful for providing feedback at scale, such as difference visualization, threaded conversations, file navigation, versioning, the ability to save drafts, and email generation. Our branch of the tool also includes a permissions model and comment sharing mechanism designed for giving composition feedback to students. Our code review tool[2] and its source code[3] are publicly available for general use.

Our purpose in using this tool was to improve the composition quality of student programs. To measure this effect,

[1] `http://code.google.com/p/rietveld`
[2] `http://composingprograms.com/codereview.html`
[3] `http://github.com/moowiz/ucb-codereview`

we asked a group of advanced undergraduate judges to compare student submissions across semesters, before and after introducing this new tool. Results of a blind and randomized survey show that quality improved by a statistically significant margin for submissions from students who received online composition feedback, compared to those from a year before when students instead received comments on printouts of their submissions.

2. RELATED WORK

Web-based feedback mechanisms have provided substantial benefit to many CS courses. Our tool shares key features of successful previous systems, such as the ability to give comments inline with the source code [5] and also allowing summary comments [10]. Because it extends the existing open-source tool Rietveld that is maintained by a large team of 16 committers, our tool also contains a breadth of additional features. For example, it integrates with Mercurial, Subversion, Git, Perforce, and CVS. Its user interface automatically collapses long stretches of matched lines. Keyboard shortcuts are implemented for file and comment navigation. Submissions are indexed and searchable through both a web form and a JSON-based REST API. Perhaps most importantly, common tasks such as deployment and uploading are fully documented on the Rietveld Wiki[4] pages.

Evaluations of past systems have shown improvements in course staff experience with the use of a web-based feedback tool. For example, MacWilliam and Malan [10] found that the total time required to provide feedback was substantially reduced with a web-based system over a PDF-based process. Likewise, Jones and Jamieson [8] report faster grading times with a web-based system. Bridge and Appleyard [2] report giving faster feedback to students.

Student experience has also been shown to improve with online submission and feedback. For instance, Bridge and Appleyard [2] report that students prefer to submit assignments online. Online feedback has also correlated with higher exam scores [5].

3. COURSE POLICIES

CS 61A has always espoused the idea that programs are a medium of communication among people; that "programs must be written for people to read, and only incidentally for machines to execute," [1]. At the same time, we believe that forcing students to adhere to a rigid and prescriptive low-level style guide would draw their attention away from the central aspects of program composition: concise implementations, clear naming, modular decomposition, and effective functional abstraction. We give our students the following guidelines to promote composition quality without suppressing their personal stylistic judgment.

3.1 Composition Guidelines

Our general composition guidelines emphasize the following aspects of program design:

Names Choosing names that evoke the purpose or meaning of the values to which they refer.

Functions Defining functions with a clear role in the program that abstract the implementation of a behavior.

Purpose Removing all unused and unnecessary code and comments, leaving behind only code with a purpose.

Brevity Preferring concise implementations to convoluted ones, without explicitly minimizing the line count.

Students were instructed that these elements together determined a program's overall composition quality. The original guidelines[5] given to students during the Fall 2012 semester in which this paper's experiments were conducted also include examples and links to further reading.

3.2 Scoring and Feedback

To provide a direct incentive for students to write comprehensible programs, we assigned a small composition score for every assignment, in addition to the correctness score assigned by our autograding scripts. A composition score indicated a coarse-grained classification of the overall quality of the submission. 12 of 94 total project points (13%) were assigned for composition across 4 programming projects.

This composition score was assigned by 10 members of the course staff, who selected those scores after writing composition feedback for each submitted project. Consistent scoring proved to be a major challenge. At the outset, we agreed on a feedback rubric for each project and discussed sample submissions in a group meeting, in order to calibrate responses. However, questions still arose among the course staff about what feedback to give for individual student responses. The features of the online tool described in the next section facilitated those discussions effectively.

4. INSTRUCTIONAL CODE REVIEW

All composition feedback was given through a web-based interface that managed the work flow of the course staff and all communication with students. A screenshot of the tool appears in Figure 1.

This section highlights the features of the tool that proved important to giving consistent, efficient, and effective feedback in a large course. These features were either available already as part of Rietveld or straightforward to add to its extensible code base, indicating a strong overlap between the use case of industrial code review and instructional feedback.

4.1 System Overview

Rietveld is a web application that manages the process of code reviews for software projects with large developer communities. Originally authored by Guido Van Rossum, Rietveld is used today by developers of the standard distribution of the Python programming language[6], the Chromium browser[7], and the Go programming language[8].

The data model groups all submissions by a student into a sequence that begins with the starter code distributed by the course staff. Students can revise their projects in response to feedback, and changes are tracked across revisions. Feedback comments and responses are each associated with a line of code in a particular submission of a project. The interface allows a user to browse the different submissions by a student and shows the feedback comments inline.

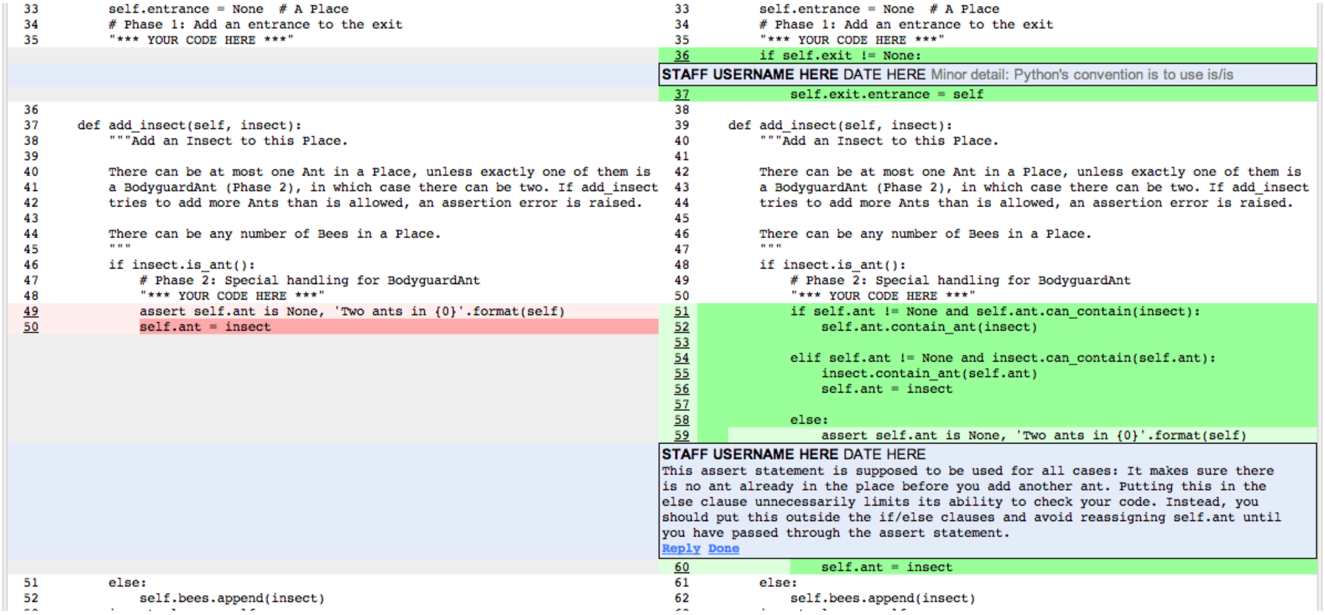
[4] https://code.google.com/p/rietveld/w/list

[5] http://inst.eecs.berkeley.edu/~cs61a/fa12/composition.html

[6] http://www.python.org

[7] http://www.chromium.org

[8] http://golang.org



```
        self.entrance = None  # A Place
        # Phase 1: Add an entrance to the exit
        "*** YOUR CODE HERE ***"

    def add_insect(self, insect):
        """Add an Insect to this Place.

        There can be at most one Ant in a Place, unless exactly one of them is
        a BodyguardAnt (Phase 2), in which case there can be two. If add_insect
        tries to add more Ants than is allowed, an assertion error is raised.

        There can be any number of Bees in a Place.
        """
        if insect.is_ant():
            # Phase 2: Special handling for BodyguardAnt
            "*** YOUR CODE HERE ***"
            assert self.ant is None, 'Two ants in {0}'.format(self)
            self.ant = insect
        else:
            self.bees.append(insect)
```

```
        self.entrance = None  # A Place
        # Phase 1: Add an entrance to the exit
        "*** YOUR CODE HERE ***"
        if self.exit != None:
```

STAFF USERNAME HERE DATE HERE Minor detail: Python's convention is to use is/is

```
            self.exit.entrance = self

    def add_insect(self, insect):
        """Add an Insect to this Place.

        There can be at most one Ant in a Place, unless exactly one of them is
        a BodyguardAnt (Phase 2), in which case there can be two. If add_insect
        tries to add more Ants than is allowed, an assertion error is raised.

        There can be any number of Bees in a Place.
        """
        if insect.is_ant():
            # Phase 2: Special handling for BodyguardAnt
            "*** YOUR CODE HERE ***"
            if self.ant != None and self.ant.can_contain(insect):
                self.ant.contain_ant(insect)

            elif self.ant != None and insect.can_contain(self.ant):
                insect.contain_ant(self.ant)
                self.ant = insect

            else:
                assert self.ant is None, 'Two ants in {0}'.format(self)
```

STAFF USERNAME HERE DATE HERE

This assert statement is supposed to be used for all cases: It makes sure there is no ant already in the place before you add another ant. Putting this in the else clause unnecessarily limits its ability to check your code. Instead, you should put this outside the if/else clauses and avoid reassigning self.ant until you have passed through the assert statement.

```
                self.ant = insect
        else:
            self.bees.append(insect)
```

Figure 1: A screenshot of the code review tool used to give composition feedback. This view is presented to a course staff member when writing feedback for a student. Red text was removed from the starter code. Green text was added by the student. Inline comments can be expanded or collapsed to a single line.

4.2 Difference Visualization

The standard view of a submission is a highlighted difference visualization between the submitted code and the starter code distributed to students. As our projects include a substantial amount of scaffolding, having a clear visualization of what was changed by the student is a boon for grader efficiency.

Difference visualizations can also be displayed for multiple submissions by the same student, so that revisions made in response to feedback can be easily tracked.

4.3 Conversation Threading

Each feedback comment begins a threaded conversation. Students are able to respond to individual comments, asking for clarification or justifying their original choices.

We did not require that students respond to comments, but we encouraged them to do so if they did not agree or wanted clarification.

4.4 Workflow Management

The tool provides instructors with a list of unfinished tasks, including ungraded assignments and responses from students that need to be addressed.

The tool tracks both students and instructors as users of the system. It assigns each student submission to a particular instructor, who is responsible for providing feedback. We chose to assign each student to the same grader for every project, so that graders could track their students' progress. In the future, we plan to experiment with a rotation of graders so that students are exposed to multiple opinions.

4.5 Permissions Control

All instructors are given permission to view all student submissions, and submissions are given unique URLs. Hence, one instructor may ask another to help review a tricky case by emailing a link. In our experience, shared access to submissions facilitated useful discussions among the course staff.

Students are only able to see their own submissions, along with any comments made by the course staff. They view the same interface as the instructors, and therefore can make comments on any line of code. In this way, they can use the tool to ask direct questions about their submissions. Shared access among the course staff facilitates auditing in the case that a student complains about the feedback that she or he received.

This permissions model is customized for instructional use. In Rietveld, all issues are public.

Extending the permissions model has proven to be straightforward. For instance, we were also able to adapt the tool to support an anonymous peer reviewing experiment wherein students commented on each other's submissions. While the details of that experiment are beyond the scope of this paper, it is worth noting here that the modular design of Rietveld allowed for substantial changes with minimal fuss.

4.6 Communication

Students receive emails from the tool when they have submitted and when they are given feedback from an instructor. The feedback is included directly in the email, along with corresponding line numbers. Thus, students do not need to log into a separate system to read their feedback.

We believe that students read a much larger portion of the feedback given to them via this email-based tool, compared to our old system of writing comments on paper and returning that paper to students. In previous years, vast stacks of unclaimed printed assignments would accumulate. As a result, the course staff was not motivated to provide helpful feedback. With this tool, all students receive their feedback and some even respond.

4.7 Repeated Feedback

Snippets are another feature we added to tailor the system to instructional use.[9] The interface for writing comments includes access to previously authored comments from the current instructor and a pool of comments shared among all instructors. Any comment can be added to this cache of snippets. In this way, members of the course staff can share their most frequent responses with each other, and consistent messages are sent to students.

This feature is particularly helpful in suggesting alternate implementations to students. One instructor can generate a good example solution to a question, and all others can suggest it to their students when appropriate.

The shared snippet memory also alerts each member of the grading staff to the types of issues encountered by others, which helps to keep the entire staff synchronized.

4.8 Configuration and Deployment

The software runs on Google App Engine, a hosting service that manages all server configuration and distributed data storage. User authentication and email addresses are managed via Google accounts. The system administration effort required is minimal.

In our institution, all student email addresses are associated with Google accounts, and so no sign-up or registration is required. Free Gmail accounts would also suffice. Extending the system to use additional forms of authentication should be feasible, although we have not tried to do so.

4.9 Version Control Integration

The tool accepts changes by reading the commit diff formats of various version control systems. Thus, students can submit their projects simply by tagging a commit to version control, then running a script to upload their submission. This upload script is maintained and distributed as part of Rietveld.

Our course does not require students to use version control. They submit via a command-line script, and we execute the necessary version control commit and upload calls to push their submissions to the online system.

4.10 Usage

The software scaled effectively to a course with more than 700 students enrolled. In the semester that we introduced the tool, a total of 8505 comments were made for 3325 project submissions. All of these comments and their responses are tracked in a database to support analysis.

5. EVALUATIONS

We developed our tool and associated course policies in order to improve the composition quality of project submissions. We evaluated our students' experience through an end-of-course survey, and we evaluated their composition quality by comparing their submissions to those of a previous semester, Fall 2011, in a blind survey.

5.1 Student Survey Results

At the end of the course, we asked students three questions related to the composition component of the course. Table 1 lists the questions and mean responses. Students answered each question using a sliding scale from 1 to 5.

Question	Female	Male	Total
How much did you read of the composition feedback that you received on your project submissions?	4.30	4.13	4.17
How helpful was the composition feedback you received?	3.60	3.37	3.42
How well did the composition feedback software system work for you?	3.92	3.71	3.75
Number of responses	134	485	626

Table 1: Mean student response to three end-of-semenster questions about the composition feedback effort. Possible responses to each question were 1 through 5 on a sliding scale (5 is better). Mean responses by gender are given for students who willingly reported their gender identity.

The majority of students report reading the feedback, with 74% responding a 4 or 5 out of 5. Those who reported their gender as *female* gave higher responses overall by a statistically significant margin ($p < 0.05$) by a Pearson's chi-squared test with four degrees of freedom.

About half of students reported that the feedback was helpful, with 49% responding 4 or 5 out of 5. Again, those students identifying as *female* gave higher responses overall, but the difference was not statistically significant.

Three fifths of students reported that the software tool was helful, with 60% responding 4 or 5 out of 5. Those students identifying as *female* gave higher responses overall, but the difference was not statistically significant.

Overall, students regarded our composition quality effort positively. In a free response survey question, some students indicated that scores for composition quality seemed arbitrary and too harsh. In the future, we intend to calibrate scores across graders, as suggested by MacWilliam and Malan [10].

5.2 Blind Survey Design

To evaluate whether composition feedback led to higher quality submissions, we performed a comparison of responses to a single question that was unchanged between Fall 2011, before our composition feedback effort, and Fall 2012, when this effort was implemented using our web-based tool. We chose this targeted setting to minimize other sources of variance across semesters.

The problem we evaluated was to implement a function to compute the centroid of a polygon, represented as a sequence of adjacent vertices. This problem appeared in their second project [3], after they had received composition feed-

[9] Snippet memories were implemented by Chenyang Yuan.

Question	Fall 2011	Fall 2012
To what extent do names introduced in the implementation convey the meaning and purpose of their values?	2.14	2.25
To what extent is their implementation concise?	1.81	2.21
Overall, to what extent has the implementation fulfilled the composition guidelines?	1.86	2.07

Table 2: Mean grader response to a blind survey comparing student submissions from semesters before (Fall 2011) and after (Fall 2012) the implementation of our composition feedback effort. Possible responses to each question were "Somewhat" (a score of 1), "Mostly" (a score of 2), or "Completely" (a score of 3).

back from their first project [12]. Students were directed to the Wikipedia page on Centroids[10] to complete their implementation. Correctness tests were provided in the project.

Submissions are mostly comparable across semesters. The course content and assignments remained largely the same. The problem wording was unchanged between years. However, the Wikipedia page describing how to compute a centroid was edited during the Fall 2012 semester: one of the students was dissatisfied with the original notation and improved it during the course of the semester. However, we don't believe that this change substantially affected student submissions, as the course forums in both semesters contained several descriptions of the computation that clarified any issues with the Wikipedia description.

We sampled uniformly without replacement 16 submissions from each year, filtering out any submissions that failed to implement the centroid function. We isolated the student code for finding a polygon centroid and removed any personally identifying information and cues about the semester from which the submission came. These submissions were then shuffled randomly into a 32-sample test set.

Each submission was evaluated by six graders from the Fall 2013 course offering. Graders have all completed the full course and received excellent overall scores and participation scores, so we consider them to be expert raters for this task. None of these graders were involved in the original composition quality effort as course staff. One of them did report that he believed his submission from the course in Fall 2012 was included in the random test set.

5.3 Blind Survey Results

These expert graders were asked to rate each submission on several dimensions, responding either "Somewhat" (score of 1), "Mostly" (score of 2), or "Completely" (score of 3). Table 2 shows their mean responses. Graders were also given a "Not at all" option, but it was never used.

Effective use of names in their implementation improved slightly in Fall 2012, but the margin of increase is not significant. A centroid computation does have intermediate values, but they don't have natural names. Hence, a wide range of arbitrary naming strategies were used by students. We cannot draw a meaningful conclusion about whether students improved their use of names based on this single problem.

On the other hand, Fall 2012 submissions were more concise by a significant margin ($p < 0.001$). A strong emphasis was placed on concise implementations in the feedback from the first project, and students appear to have responded by submitting more direct implementations rather than convoluted ones.

Overall, submissions in Fall 2012—the semester that included composition feedback—were judged to have better fulfilled the composition guidelines described in Section 3.1. This difference is statistically significant ($p < 0.01$) by a Pearson's chi-squared test with two degrees of freedom.

6. CONCLUSION

We introduced a new tool to facilitate feedback at scale, based on an existing open-source project. We used the tool to place renewed emphasis on the composition quality of student submissions. In the semester that we began providing feedback with this tool, the composition quality of student submissions improved.

We have confirmed many of the observations of previous similar efforts in other courses, as well as outlined some of the features of our tool that proved particularly effective in a large course. We look forward to improving the tool based on the feedback of other users in the computer science community.

7. REFERENCES

[1] H. Abelson and G. Sussman. *Structure and Interpretation of Computer Programs.* MIT Press McGraw-Hill, Cambridge, Mass. New York, 1985.

[2] P. Bridge and R. Appleyard. A comparison of electronic and paper-based assignment submission and feedback. *British Journal of Educational Technology*, 39(4):644–650, 2008.

[3] J. DeNero and A. Muralidharan. The twitter trends project. In *Nifty Assignments Track of SIGCSE*, New York, NY, USA, 2013. ACM.

[4] S. H. Edwards and M. A. Perez-Quinones. Web-CAT: automatically grading programming assignments. In *Proceedings of the 13th annual conference on Innovation and technology in computer science education*, ITiCSE '08, pages 328–328, New York, NY, USA, 2008. ACM.

[5] D. Heaney and C. Daly. Mass production of individual feedback. In *Proceedings of the 9th annual SIGCSE conference on Innovation and technology in computer science education*, ITiCSE '04, pages 117–121, New York, NY, USA, 2004. ACM.

[6] J. Hollingsworth. Automatic graders for programming classes. *Communications of the ACM*, 3(10):528–529, Oct. 1960.

[10] `http://en.wikipedia.org/wiki/Centroid`

[7] D. Jackson and M. Usher. Grading student programs using assyst. In *Proceedings of the twenty-eighth SIGCSE technical symposium on Computer science education*, SIGCSE '97, pages 335–339, New York, NY, USA, 1997. ACM.

[8] D. Jones and B. Jamieson. Three generations of online assignment management. *What works and why : reflections on learning with technology : ASCILITE*, 1997.

[9] D. Knuth. *Literate Programming*. Center for the Study of Language and Information, Stanford, Calif, 1992.

[10] T. MacWilliam and D. J. Malan. Streamlining grading toward better feedback. In *Proceedings of the 18th ACM conference on Innovation and technology in computer science education*, ITiCSE '13, pages 147–152, New York, NY, USA, 2013. ACM.

[11] D. Morris. Automatic grading of student's programming assignments: an interactive process and suite of programs. In *Frontiers in Education, 2003. FIE 2003 33rd Annual*, volume 3, pages S3F–1–6 vol.3, 2003.

[12] T. Neller. Pig. In *Nifty Assignments Track of SIGCSE*, New York, NY, USA, 2010. ACM.