

Workshop #110: Making Manual Code Review Scale

Preamble

This walkthrough has two goals:

- 1) To explain the basics of using codePost to run a course
- 2) To explain how to implement the personalized workflow strategies discussed during the first part of this workshop.

To demonstrate how codePost works in a realistic setting, we'll all use the following sample assignment, called [The Birthday Problem](#) (exam written by Bob Sedgewick). Java is our language of choice, though none of the steps we'll walk through together will be specific to Java. They'll work just as well with Python, C, or any other common language.

Part 1: codePost basics

Step 0: Creating an account

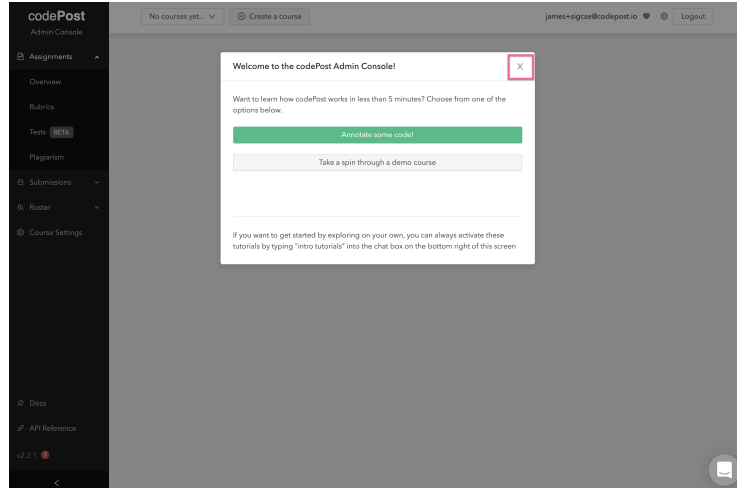
To create a codePost account, navigate to <https://codepost.io/signup/create>. From here, you can create an **admin** account, giving you the ability to create a course and manage it.

Soon after signing up, you'll receive an email from codePost. This email will contain a link, which you can follow to set your codePost password.

Now, login to codePost from <https://codepost.io/login>.

Step 1: Creating a course

After logging into codePost for the first time. You'll see a page that looks like this.



Close the pop-up window by clicking the x on the top right. Then create a course by clicking the “Create a course” button at the top of the page.

You can give the course whatever “name” and “period” you like.

A screenshot of the "Create a course" form. The form has a title "Create a course" and a close button (X) in the top right corner. It contains two required fields: "Course name" with the value "SIGCSE Demo" and "Course period" with the value "2020". At the bottom right of the form are two buttons: "Cancel" and "Create".

Step 2: Adding a roster

After you’ve created a course, the next step is to add **students** to your course. To do this, click the “Roster” button on the left side of the page, then hit “Students.” From here, click “Add Students”.

For the purposes of this demo, we’re going to add some dummy students. codePost uses emails to index students. Copy and paste the list below into the textbox you see:

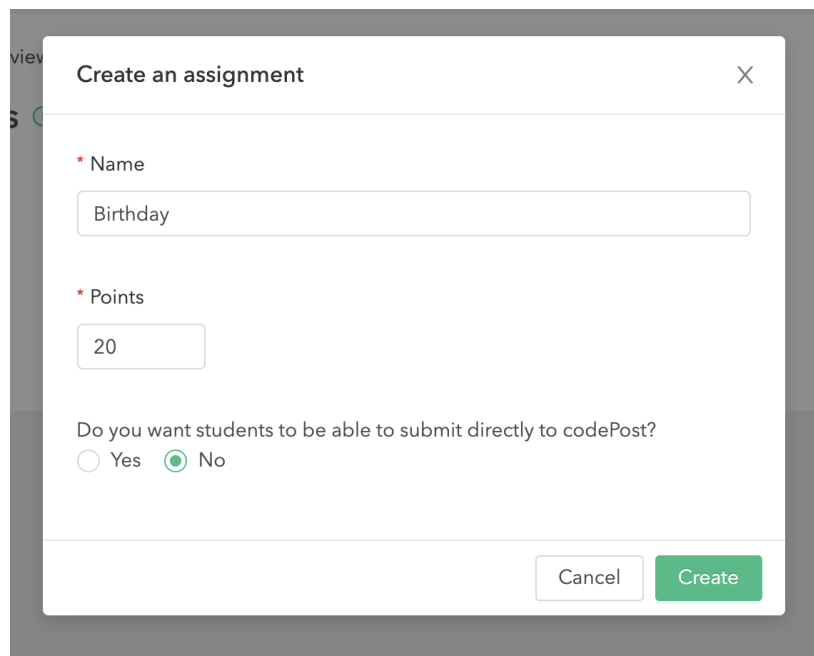
student0@codepost.io

student1@codepost.io
student2@codepost.io
student3@codepost.io
student4@codepost.io

Then click “Review Roster Changes” and finally “Confirm”. Now your course has 5 students! Once added to a course in codePost, students can log in to submit work and view feedback and scores (once grading is complete).

Step 3: Creating an assignment

Next, we’re going to create an assignment. To do this, head to “Overview” tab you see under “Assignments” on the left side of the page. Then click “Add assignment.”



The screenshot shows a modal dialog titled "Create an assignment" with a close button (X) in the top right corner. The dialog contains the following fields and options:

- Name:** A text input field containing the text "Birthday".
- Points:** A text input field containing the number "20".
- Submission Option:** A question "Do you want students to be able to submit directly to codePost?" with two radio buttons: "Yes" (unselected) and "No" (selected).
- Buttons:** "Cancel" and "Create" buttons at the bottom right.

Call it “Birthday” and give it 20 points. Since we don’t have real students in our course, you can answer “No” to the question at the bottom of the window.

Step 4: Uploading submissions

Now that students are added and an assignment has been created, we can upload submissions to codePost. In many course configurations, instructors choose to have students submit directly to codePost. But instructors can also upload submissions manually. Since our students aren’t real, we’ll use the latter option.

[Click here to download](#) a zipped containing 5 submissions for the Birthday assignment, representing sample submissions from our 5 students.

Once downloaded, unzip the file. Click the button in the “Actions” column of the “Birthday” assignment. From here, select “Upload submissions” and then “Multiple submissions.” Now, drag the folder you just downloaded into the pop-up. Follow the instructions in the pop-up to upload these submissions.

Step 5: Writing simple tests

Next, we’ll create some automated tests for the Birthday assignment.

Navigate to the “Tests” tab and hit “Edit.” From here, select “Java” as your language. Doing so will create a container for your assignment. All code you run (test code and student code) will run within this container.

Since this is Java, we’ll need to make sure the student code gets compiled. That’s the job of **runscript**. The runscript code will run prior to any test code. For Java, codePost supplies a default runscript that will compile any Java code present in a student submission.

Finally, before we start writing tests, we’re going to add two files to help us with the test writing process:

- Helper file: these files are accessible to our tests. Upload [20birthdays.txt](#), which is a sample input for the Birthdays program.
- Solution code: a working solution. We’ll use solution code to test our tests, ensuring they function as we expect. Upload [Birthday.java](#).

Now we’re ready to create our tests. First we’ll go over how to create basic tests in codePost, and then we’ll create the ‘Level 1’ requirements.

Hit the “Tests” and then click “Add category.” codePost **Test Cases** are organized into **Test Categories**. Let’s call this category “Correctness”.

Now click “Add test” to create a test case. Call this test “Test 1”. For this first test, we want to run the student’s code with the command `java Birthday 2 20birthdays.txt` and ensure that we get the expected answer: `7 birthdays examined to find 2 occurrences of 101`

For this test, we’re going to use codePost’s “Input / Output” test type. This is the simplest type of test, used to assert certain outputs from specific method calls.

Since we’re going to be testing the main (and only) method of Birthday.java, we’ll choose “from **command line**” instead of the default “from **file**”.

Next, fill in the two inputs with the command you want to run and the output you expect. When you're done hit **Run** in the top right of the terminal at the bottom of the screen. This will run your test on solution code. If your test is configured correctly, the test should pass!

If you want to get fancy, you can specify a regular expression to compare against the student's output. For example, you could use the following regexp to ensure that the student code outputs a string containing first a 7, then a 2, then a 101: `7([0-9]+)2([0-9]+)101`

2. Definition

From file **command line** run the command

and expect the call to print the value equal to .

Now, create the following two tests in the same way.

- Confirm that `java Birthday 3 20birthdays.txt` **outputs** `17 birthdays examined to find 3 occurrences of 123`
- Confirm that `java Birthday 4 20birthdays.txt` **outputs** `No birthday occurs 4 times`

Now that we have these simple tests set up, we can run them on the students code. Hit "View Results" in the top right of the test editor. Then hit "Run all tests." This will run the 3 tests you created on the 5 student submissions you uploaded earlier.

Part 2: personalized feedback in codePost

2.0: Creating "Level 1" requirements

As discussed, a potent strategy for ensuring students write reviewable code is to create some "Level 1" requirements against which students can check their code.

Having built some basic input / output tests in the last section, we're ready to write our Level 1 tests. First, we're going to write a test that checks to make sure the student code compiles.

To do this, create a new test category called "Level 1." Create a new test in this category. Call this test "Compile check." For the test type, instead of "Input / Output", select "Unit Test (Bash)".

Unit tests allow you to write small snippets of code to execute a test. Within a bash unit test, your code must call the command `TestOutput` (which is a custom `codePost` command). `TestOutput` takes two arguments: `<passed: true/false>` and `<logs: string>`.

All we need to do to make sure the student code compiles is to run `javac Birthday.java`. If the command is successful, we want the test to pass, and fail otherwise. The following bash command does the trick: `javac Birthday.java && TestOutput true "Compiled!" || TestOutput false "Compilation failed"`

Next, we're going to write a test to confirm that the student's `Birthday.java` program contains a main method. To do this, create a new test and select "Unit Test (java)" as your test type.

Writing a unit test in Java is very similar to writing a unit test in Bash. Instead of calling the `TestOutput` *command*, Java unit tests have to *return* a `TestOutput` *object*. The `TestOutput` object is constructed with two arguments: `<passed: true/false>` and `<logs: string>`.

You can write this unit test however you'd like. Below is a code snippet that will do the trick, using [reflection](#).

```
import java.lang.reflect.Method;

class Test {
    static TestOutput Test() {

        Birthday bday = new Birthday();
        Method[] methods = bday.getClass().getMethods();

        boolean hasMethod = false;
        for (Method m : methods) {
            if (m.getName().equals("main")) {
                hasMethod = true;
                break;
            }
        }

        if (hasMethod) {
            return new TestOutput(true, "main method found!");
        } else {
            return new TestOutput(false, "no main method");
        }
    }
};
```

If this were a real course and students were submitting code to codePost, you would need to “expose” this tests to make them visible to students when they submit. To do that, hit “More options”, check “Exposed”, and then save. Here’s a video of a student submitting to codePost and running two tests like the ones we just wrote.

[This video](#) shows what it would look like for your students to submit to an assignment.

Before moving on to the next step, make sure you re-run all of your tests to ensure the 2 new tests you wrote are executed.

2.1: “Tag then explain” using codePost explanations

Now we’ll explore the “Tag then explain” technique and also review how code review works in codePost.

Go back to the Assignments Overview page using the tab on the left of the Admin Console. From here, click the “5” under the Submissions column. From here, click on one of the student emails to open a submission.

The submission will open in the codePost [Code Console](#). This is where code annotation happens. To create a comment, try highlighting some code with the mouse. You can use plaintext or Markdown in comments.

Let’s practice “Tag then explain.” To do this, we’ll learn how to:

- Create a rubric comment
- Apply a rubric comment
- Write a rubric comment explanation

To create a rubric comment, hit the green pen icon in the rubric section on the left. Rubrics, like tests, are two-tiered. First, create a category called “Style.” Now, add a comment called “magic numbers.” Save the rubric before continuing.

This comment will be our tag. Let’s apply the tag to the submission we currently have open. Do you see any magic numbers in the student code?

To apply a rubric comment, create a comment by highlighting some code, and then click the rubric comment to apply it to the highlighted code.

Now, go through the other submissions and tag instances of magic numbers using this rubric comment.

Once you're done, head back to the Admin Console. Click "Rubrics" on the left and then "Create." This is the rubric editor. You should see the comment you made in the Code Console, along with a number showing how many times you applied the comment.

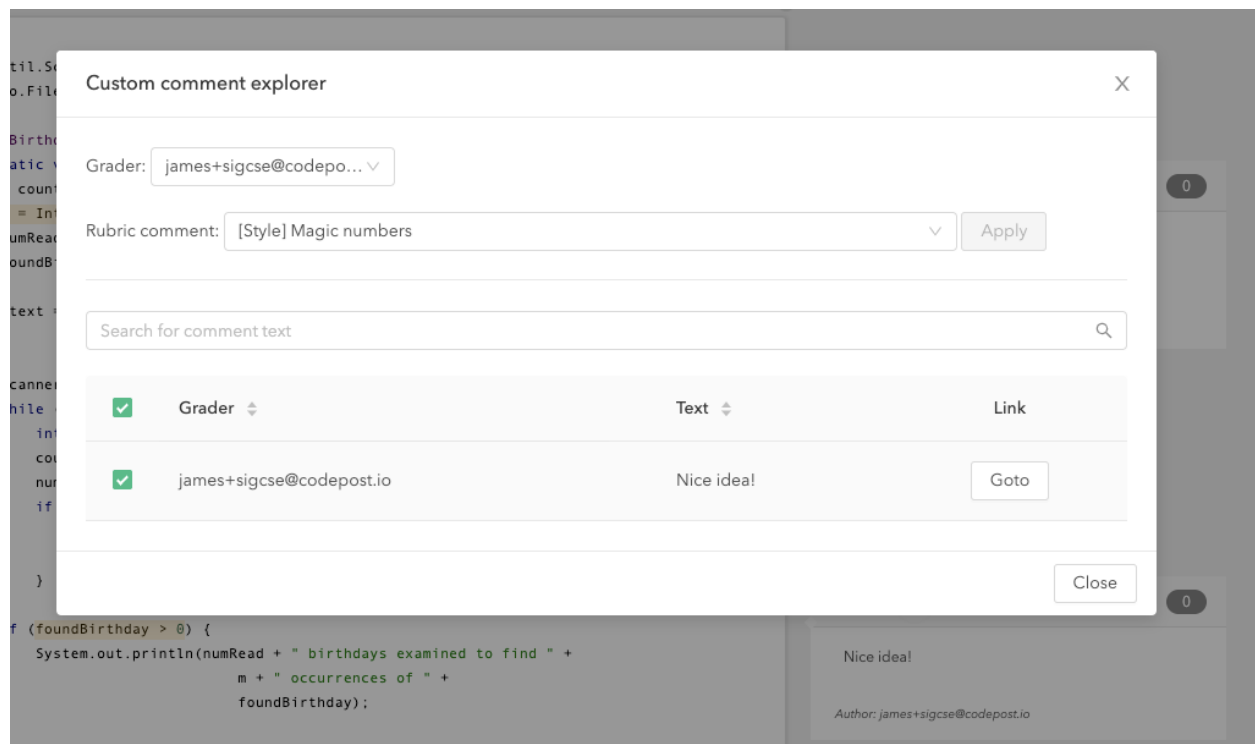
To create an explanation, enable the "Show explanation editor" setting. Then click the pen icon in the "Explanations" tab. From here, you can create a thorough explanation that your students will see instead of just "Magic numbers."

2.2: Building rubric iteratively

codePost includes functionality to allow you to retroactively tag a comment with a rubric comment. As a leader instructor, this allows you to do two things:

- (a) Ensure a high percentage of your comments leverage rubric comments
- (b) Audit the work of your graders to ensure usage of the rubric

To tag a comment with a rubric comment, hit cmd-k (Mac) or ctrl-k (Windows), and search for "custom comment explorer." Hit return to open it.

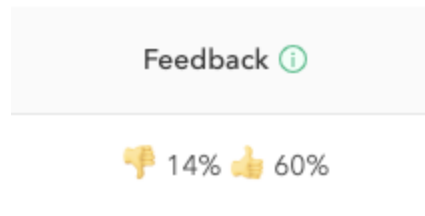


2.3: Turning on student feedback

Student feedback is turned on by default. To verify, open up a submission and then click the icon to the left of the finalized toggle ("view as student"). This will show you what the submission looks like from the POV of your students. If student feedback is enabled, you'll see a "thumbs

up” and “thumbs down” button underneath the rubric comment. Students use these buttons to indicate whether or not they found the comment helpful.

Votes aren't shown at the student-level; in other words, you can't see whether a particular student found a rubric comment helpful. Instead, scores are aggregated in the rubric editor.



2.4: Adding graders

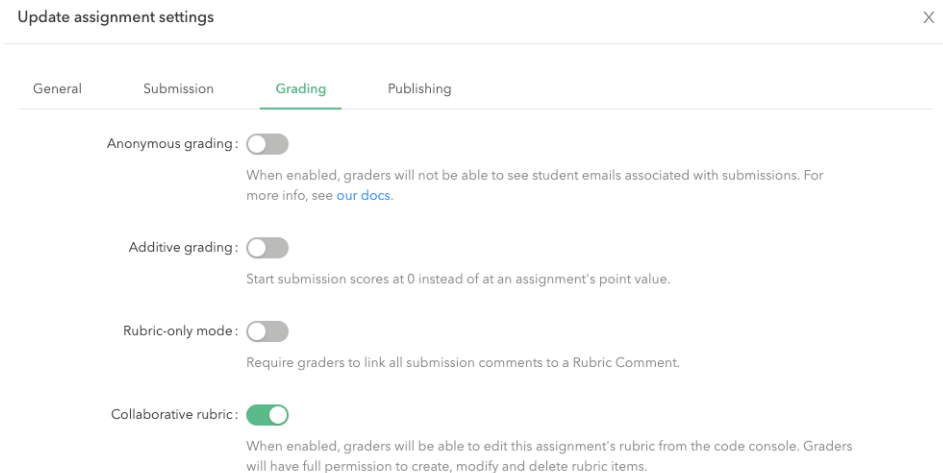
Before discussing any of our multi-threaded strategies, a note on adding graders to your course. In codePost, graders are a distinct class of user. They have the ability to review submissions, but don't have full read/write access to a course, like admins do.

Adding graders to your course is just like adding students, done from the “Graders” tab under “Roster” in the Admin Console.

To see what the interface for Graders looks like, you can go there by clicking the codePost logo in the top left of the page and then selecting “Grader Console.”

2.5: Collaborative rubric creation

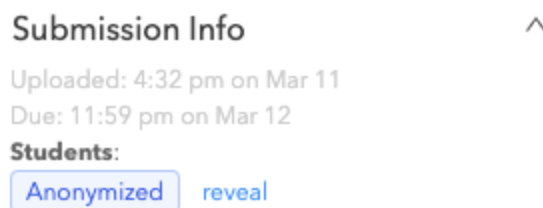
Turning on collaborative rubric creation is easy. Just navigate to an assignment's settings from the Assignment Overview page. Select “Grading” and then toggle on “Collaborative rubric.”



Now, graders will be able to edit a rubric from the Code Console, just like we did earlier.

2.6: Anonymous grading mode

Anonymous grading mode is another setting. Try turning it on and then opening up a submission. In the top left of a submission, you should see this:



Clicking reveal will show the identity of the student. Note if this setting is on, graders won't have the ability to reveal student identities.

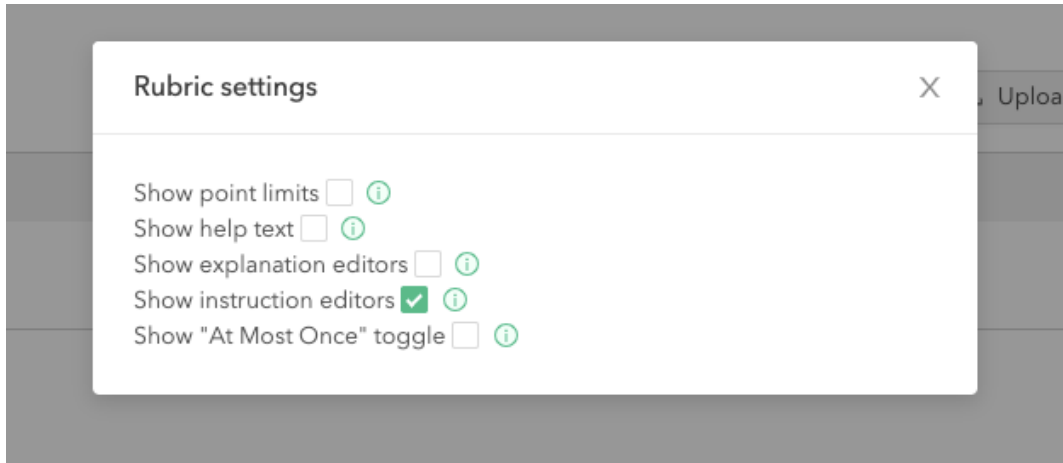
2.7: Enforcing quality: rubric-only mode and instruction text

We discussed two techniques for enforcing high quality code reviews:

- Instruction text
- Rubric-only mode

To enable rubric only mode, go back to the “Grading” tab of your assignment settings and turn on “Rubric-only mode.” Once enabled, custom comments without attached rubric comments will be disallowed.

Instruction text can be set from the Rubric Editor. Go there, and then open up your settings and turn on “Show instruction editors”.



Once enabled, you can add instructional text to any rubric comment. If instruction text is used as template text, then any comments which apply the rubric comment will have their custom text pre-filled with instruction text. This provides a strong nudge to graders, and is useful for “fill-in-the-blank” style instruction text.

