

EECS481 HW6b - Project Report

1. Names and Email IDs

This is a joint project report by Shuta Suzuki (shutas) and William Dempsey (wdempsee).

2. Selected Project

We selected Oppia (pronounced "O-pee-yah", which is Finnish for "to learn"), an open-source, online educational platform provided at no cost. It allows anyone to "create and share interactive activities," and it offers a wide range of lessons on topics from math and sciences to foreign languages and humanities. At a high level, the project is divided into frontend and backend. Frontend includes all of the relevant code for the client-side website code, primarily written in HTML, CSS, and JavaScript. Backend consists of server code which takes the users' requests and processes them, as well as platform dependent code which provide interfaces of third-party services Oppia uses (e.g. Google App Engine, ndb, etc.). Both components of the backend are written in Python.

Oppia: <https://www.oppia.org/splash>

GitHub: <https://github.com/oppia/oppia>

3. Project Context

First publicly announced in 2014, Oppia originally started as one of the projects by Google Open Source. As part of Google's mission to use technology to reimagine the way we learn, Oppia began offering high-quality, interactive lessons which incorporate videos, images, and open-ended questions designed to stimulate intellectual curiosity. Currently, Oppia is owned by the Oppia Foundation, a non-profit organization based in California "to provide high quality education to those who lack access to it." Since the start of the project, Oppia has provided free lessons to "1 million learners in over 140 countries." Oppia is similar to other online education tools such as Udemy (<https://www.udemy.com/>) and Coursera (<https://www.coursera.org/>), however remains unique in that it is completely free to use and allows for users to create the lessons.

4. Project Governance

There are two platforms that contributors use to communicate with each other: GitHub and Gitter. Like many open-source projects, GitHub is used as the primary communication platform once a pull request has been issued. Under each issue, the issuer of the pull request discusses the potential changes with a project owner and the developer who opened the issue. This is done via (often long) chains of comments, where you can directly reference code snippets from the codebase.

Gitter is an online instant messaging platform which is also used by Oppia's developer community. Gitter provides a dedicated chatroom for Oppia developers, and it is primarily used to communicate with other developers for hints and ideas regarding code implementation. Unlike GitHub's pull request threads, this platform allows developers to discuss about code snippets prior to making a pull request. Furthermore, many people often ask more general questions such as how to run and test certain parts of the codebase. The use of GitHub and Gitter are strictly informal, and there are no guidelines on how the communication must take place. For example, you can submit a pull request and have your changes approved without using Gitter at all. However, each pull request on Github must follow a certain formula called a "checklist," which describes how a pull request should be laid out, what information it should include, who should be assigned as a reviewer, etc.

The overall acceptance process for Oppia works as follows: submit a pull request to an existing issue in the tracker, fix any errors caused by any of the five continuous integration tests/builds (four tests via Circle CI, one build via Travis CI), and get approval by at least one of the project owners. After successful approval, the proposed changes are merged into the Oppia's production branch.

Oppia's developer community defines a general style guide for Python, JavaScript, and CSS code. Developers are encouraged to follow this predetermined style to reduce "unnecessary back-and-forth during code review." Beyond this, every line of code submitted as part of a pull request must pass static linters (e.g. pylint for Python code etc.) and integration tests, otherwise, submitted code is automatically rejected. Furthermore, there are suggested guidelines for naming variables and labeling pull requests which are not strictly enforced, but recommended (project owners will let you know when these can be improved for future contributions).

Gitter (Oppia chat page): <https://gitter.im/oppia/oppia-chat>

Oppia's Coding Style Guide: <https://github.com/oppia/oppia/wiki/Coding-style-guide>

5. Task Description

Task 1 (Backend Test)

Description: Raise the code coverage of one of the backend's platform-dependent code (`core.storage.question.gae_model`) to 100%.

Implementation: We added a new test case that uses mocking to replace a dependent method used in the `create` function for the `QuestionModel` class. This new test case simulates an unlikely scenario where the ID generator generates 10 hashes in a row, all of which causes a collision with previously generated ID hashes.

Task 2 (Backend Test)

Description: Raise the code coverage of one the backend's server code (`core.domain.topic_services`) to 100%.

Implementation: We added several test cases for various functions in `topic_services.py`. Test cases spanned from testing for raising exceptions under certain circumstances to determining whether a user had specific rights to edit “topics.” One specific issue we ran into with this set of tests is how to test “protected functions” without directly calling them, as this would trigger an error while linting the Python backend tests. In order to workaround this issue, we implemented tests that called public functions within the same class as the protected function that also called the protected functions. This allowed us to test these protected functions without actually having to call them directly in our test. Due to current version constraints, we were *not* able to raise coverage to 100%, but were able to raise coverage to 97%.

6. Submitted Artifacts

Task 1 (Backend Test)

The artifact for this task was a new test case that was added to increase the code coverage of `core.storage.question.gae_model` to 100%.

A screenshot of a GitHub pull request interface. The repository is 'oppia / oppia'. The pull request title is 'Fix part of #6550: Raise code coverage of core.storage.question.gae_models to 100% #6558'. It shows that the pull request is merged. The diff view shows changes to the file 'core/storage/question/gae_models_test.py'. The code includes imports for 'state_domain', 'platform', and 'models', and defines several test methods like 'test_create_question', 'test_raise_exception_by_mocking_collision', and 'test_apply_change_list_with_update_additional_story_id'.

File Changed: <https://github.com/oppia/oppia/pull/6558/files>

Task 2 (Backend Test)

The artifact for this task was several new test cases that were added to increase the code coverage of `core.domain.topic_services.py` from 89.7% to 97%.

A screenshot of a GitHub pull request interface. The repository is 'oppia / oppia'. The pull request title is 'Fix part of #6550: Raise code coverage of core.domain.topic_services to 91% #6628'. It shows that the pull request is open. The diff view shows changes to the file 'core/domain/topic_services_test.py'. The code includes imports for 'user_services', 'platform', 'models', 'feconf', and 'TopicChange'. It defines test methods such as 'test_assign_roles_with_invalid_new_role', 'test_get_all_topic_rights', and 'test_apply_change_list_with_update_additional_story_id'.

File Changed: <https://github.com/oppia/oppia/pull/6628/files>

7. QA Strategy

Perhaps the most obvious QA activity we performed was **code review**. For each task we implemented and made a pull-request for, we reviewed the changes with one of the project owners and made changes to our submitted code per their requests. This activity was enforced and necessary in order to have any code merged.

With every push to the Git repository, Oppia runs several QA procedures. One such procedure is **linting**. In order for our code to successfully push to the repo, it must pass several linting checks including Python linting (this was most important to us as all of our tasks were in Python). The metric for passing the Python linting is to have a perfect 10/10 Pylint rating. Like code reviews, linting is enforced by the Oppia acceptance procedure and was thus a necessary step in having our code successfully push and eventually be merged.

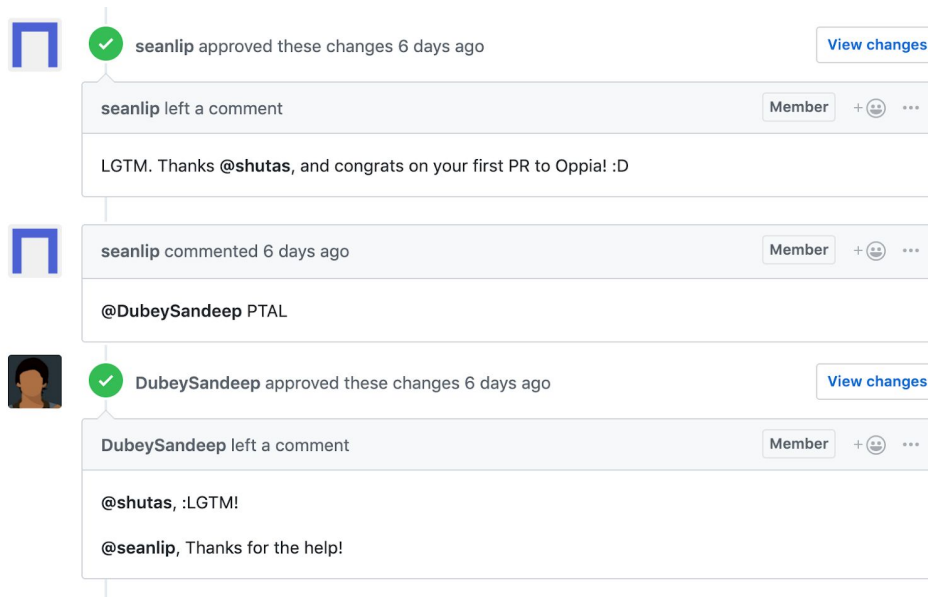
In order to have a pull-request merged into the production codebase, it must first pass several **continuous integration** builds. Four of the CI builds are done via Circle CI and consist of standard tests such as front-end and back-end tests. The last of the CI builds is a Travis CI build. As with code reviews and linting, CI builds are an automated feature of the Oppia acceptance procedure so it was absolutely necessary our code pass these checks.

Like most projects, one of the more involved QA activities we performed was **testing**. Our first two tasks were completely wrapped up in this as they sought to increase backend code coverage. The metric for these tasks was increasing **code coverage** to 100%.

8. QA Evidence

Task 1:

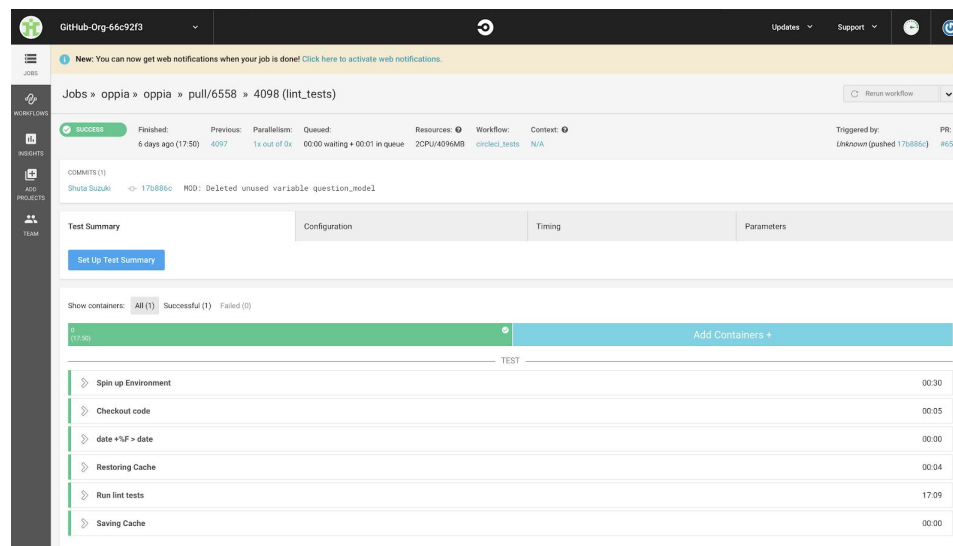
- Implemented tests: <https://github.com/oppia/oppia/pull/6558/files>
- Code review:



The screenshot shows a GitHub pull request interface with three comments:

- seanlip approved these changes 6 days ago** (with a green checkmark icon). A "View changes" button is visible to the right.
- seanlip left a comment** (Member status). The comment text is: "LGTM. Thanks @shutas, and congrats on your first PR to Oppia! :D".
- seanlip commented 6 days ago** (Member status). The comment text is: "@DubeySandeep PTAL".
- DubeySandeep approved these changes 6 days ago** (with a green checkmark icon). A "View changes" button is visible to the right.
- DubeySandeep left a comment** (Member status). The comment text is: "@shutas, :LGTM!" and "@seanlip, Thanks for the help!".

- Travis CI build: https://travis-ci.org/oppia/oppia/builds/519924675?utm_source=github_status&utm_medium=notification
- Linting: https://circleci.com/gh/oppia/oppia/4098?utm_campaign=vcs-integration-link&utm_medium=referral&utm_source=github-build-link



The screenshot shows a Travis CI build log for a successful job. The job is named "lint_tests" and is part of a pull request (PR) #6558. The build status is "SUCCESS".

Job Details:

- Finished: 6 days ago (17:50)
- Previous: 4097
- Parallelism: 1x out of 0x
- Queued: 00:00 waiting + 00:01 in queue
- Resources: 2CPU/4096MB
- Workflow: circleci_@tests
- Context: N/A
- Triggered by: Unknown (pushed 17b886c)
- PR: #6558

COMMENTS (1):

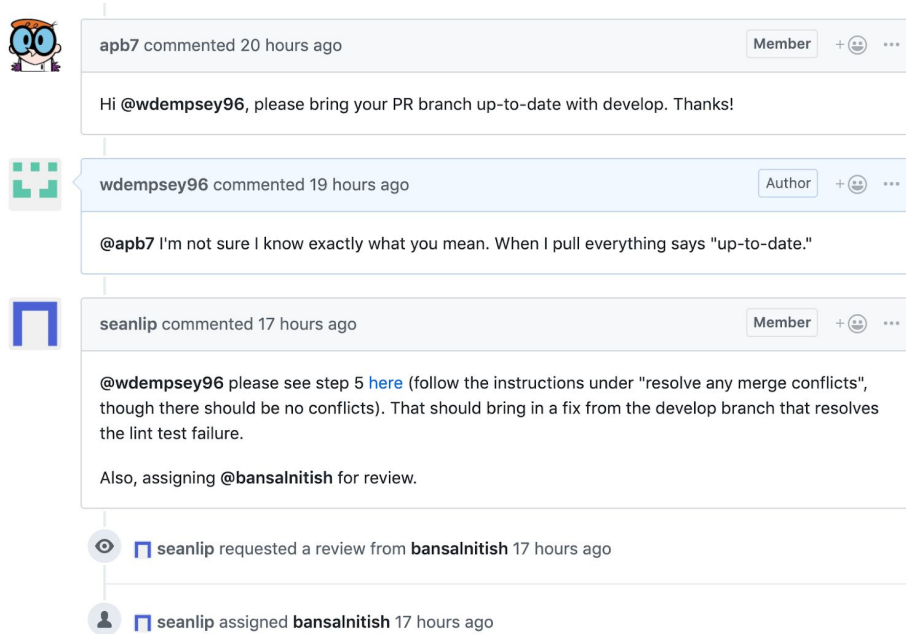
- Shuta Suzuki <@17b886c> NOD: Deleted unused variable question_model

Test Summary:

Step	Duration
Spin up Environment	00:30
Checkout code	00:05
date +%F > date	00:00
Restoring Cache	00:04
Run lint tests	17:09
Saving Cache	00:00

Task 2:

- Implemented tests: <https://github.com/oppia/oppia/pull/6628/files>
- Code Review:



The screenshot shows a GitHub pull request conversation. It starts with a comment from user 'apb7' (Member) 20 hours ago: "Hi @wdempsey96, please bring your PR branch up-to-date with develop. Thanks!". This is followed by a response from 'wdempsey96' (Author) 19 hours ago: "@apb7 I'm not sure I know exactly what you mean. When I pull everything says 'up-to-date.'". Then, 'seanlip' (Member) comments 17 hours ago: "@wdempsey96 please see step 5 here (follow the instructions under 'resolve any merge conflicts', though there should be no conflicts). That should bring in a fix from the develop branch that resolves the lint test failure. Also, assigning @bansalnitish for review." Below the comments, there are two actions: 'seanlip requested a review from bansalnitish 17 hours ago' and 'seanlip assigned bansalnitish 17 hours ago'.

- Travis CI build:
https://travis-ci.org/oppia/oppia/builds/522377530?utm_source=github_status&utm_medium=notification
- Linting: Local lint check prior to push

```
PROBLEMS 9 OUTPUT DEBUG CONSOLE TERMINAL 1: git

Linting HTML files.
Using config file /Users/Will/semester_W19/EECS-481/hw6/opensource/oppia/.pylintrc
Using config file /Users/Will/semester_W19/EECS-481/hw6/opensource/oppia/.pylintrc
Summary of Errors:
-----

SUCCESS Js component name and count check passed
SUCCESS Directive scope check passed
SUCCESS Controller dependency line break check passed

(942 files checked, 0 errors found)
SUCCESS HTML directive name check passed
SUCCESS Import order checks passed
SUCCESS Docstring check passed
SUCCESS Comments check passed
SUCCESS HTML tag and attribute check passed

SUCCESS HTML linting passed
HTML linting finished.

(1951 files checked, 0 errors found)
SUCCESS Pattern checks passed
SUCCESS Copyright notice check passed

-----
All Checks Passed.

Using config file /Users/Will/semester_W19/EECS-481/hw6/opensource/oppia/.pylintrc
CSS linting finished.
Using config file /Users/Will/semester_W19/EECS-481/hw6/opensource/oppia/.pylintrc
Using config file /Users/Will/semester_W19/EECS-481/hw6/opensource/oppia/.pylintrc
Using config file /Users/Will/semester_W19/EECS-481/hw6/opensource/oppia/.pylintrc
Using config file /Users/Will/semester_W19/EECS-481/hw6/opensource/oppia/.pylintrc
Using config file /Users/Will/semester_W19/EECS-481/hw6/opensource/oppia/.pylintrc
Python linting finished.
```

9. Plan Updates

The most significant mistake we made in our original planning was underestimating the amount of time necessary for tasks 1 and 2. We had originally scheduled time for a third task, however this underestimation ultimately led us to *dropping* task 3 all together.

In our original plan we devoted the same schedule and time to tasks 1 & 2 because they are the same task (writing test cases) for different files. However, we quickly learned that due to the additional coverage necessary in task 2 (13% to raise task 2 to 100% vs. 99% to raise task 1 to 100%), task 2 was most likely going to require more time to complete.

Because of the additional time task 2 required, we decided to utilize pair programming to implement this task (an activity we originally planned on using for task 3). This meant that our original plan of implementing tasks 1 and 2 separately was also changed in order to account for the time and effort underestimation on these tasks.

The largest deviation from the original plan for Task 2 was the actual ability to raise coverage to 100%. Due to version constraints of the current release of Oppia, 100% coverage was not possible for this task, and instead we were able to raise it to 97%.

Project Schedule:

**Note: Our original report broke down our plan by date, however here we break it down by hours in order to be more precise. Furthermore, times listed below may overlap. For example, hours of writing tests in response to project owner requests coincide with hours collaborating with project owners.*

Task 1:

- Analyze relevant parts of the codebase: *3 hours*
 - *2 hours* reviewing Oppia-provided documentation for writing backend test cases and generating coverage reports
 - *1 hour* reviewing the current test suite, the file to be tested, and files that the target file depends
 - The amount of time necessary for this individual task was inline with our original estimation.
- Write tests: *7 hours*
 - *1 hours* writing original set of test cases
 - *6 hours* writing test cases in response to requests from project owners during code review
- Generate & Review Coverage: *4 hours*

- *4 hours* waiting for coverage generation and review relevant changes in coverage report
- PR & Integration tests: *2 hours*
 - *2 hours* resolving Python linting errors during integration tests
- Collaborating with Project Owners to get PR accepted: *10 hours*
 - *9 hours* to write tests in response to project owners' requests
 - *1 hour* of other communication regarding code style and discussion of specific implementation

Task 2:

- Analyze relevant parts of the codebase: *6 hours*
 - *2 hours* reviewing Oppia-provided documentation for writing backend test cases and generating coverage reports
 - *4 hours* reviewing the current test suite, the file to be tested, and files that the target file depends
 - The amount of time necessary for this individual task is accurate to our original estimation.
- Write tests: *11 hours*
 - *3 hours* writing original set of test cases
 - *8 hours* writing test cases in response to requests from project owners during code review
 - The amount of time necessary for this individual task was longer than our original estimation.
- Generate & Review Coverage: *3 hours*
 - *3 hours* waiting for coverage generation and review relevant changes in coverage report
 - The amount of time necessary for this individual task is accurate to our original estimation.
- PR & Integration tests: *4 hours*
 - *4 hours* resolving Python linting errors during integration tests
 - The amount of time necessary for this individual task was much longer than our original estimation.
- Collaborating with Project Owners to get PR accepted: *9 hours*
 - *8 hours* to write tests in response to project owners' requests
 - *1 hour* of other communication regarding build success and other miscellaneous topics.

- The amount of time necessary for this individual task was longer than our original estimation.

10. Our Experiences and Recommendations

Task Selection:

In order to start working towards contributing to an open-source project, we first had to actually choose a project. Choosing Oppia proved to be a great choice due to the wide availability of “first-time contributor” tasks, active communication in the Oppia-contributor community, and huge amount of documentation. Oppia offers documentation on everything from forking the repo and creating a branch to implement your changes on to generating coverage reports to resolving merge conflicts in a pull-request. Because the plentiful documentation made getting started quick and easy, with minimal communication with the project owners, we highly recommended looking for well-documented projects to contribute to.

Effort Estimation and Planning:

As we have explained in the Plan Updates section, we have greatly underestimated the amount of time needed to complete tasks 1 and 2, leading us to drop task 3 altogether. One of the main reasons for this underestimation was due to the fact that code comprehension was significantly harder and time-consuming than we expected. It was difficult because many classes/modules we were writing test cases for did not have inline comments, so we were forced to read each line carefully and reason through the logic from scratch (there are no explicit guidelines for inline comments, so some developers get away with writing little or no comments beyond function docstrings). Furthermore, some modules we were testing were low-level interface modules which took us a longer time to investigate how they fit in the overall project. Code review on GitHub was also more time-consuming than we initially thought, since project owners seemed to have very high standards for any proposed changes to the production code (from new variable names to overall structure of individual test cases), so the discussion with project owners alone spanned over a week.

Perverse Incentives and Incomplete Requirements in Testing:

The topic of perverse incentives was frequently mentioned throughout the course and was one of the topics we discussed while working on tasks 1 and 2. Tasks 1 and 2 were raising coverage of certain backend files to 100%. However, the task description fails to mention any stringent testing guidelines outside of raising coverage to 100%. These incomplete task requirements lead to a perverse incentive in writing test cases that simply cover the necessary lines without actually testing anything. In other words, one

can write test cases that don't actually assert anything, which is typically the standard pattern for unit testing, but still manage to generate coverage. This leads back to the topic discussed in class that although code coverage is a very popular metric, it can be a false sense of security if used wrong.

Code Inspection and Review for Pull Requests:

The most time consuming and perhaps most important QA activity we utilized was code review. Participating in code reviews with the project owners was extremely helpful in understanding the task scope in the project, understanding and debugging any issues that come up in integration tests, and understanding the Oppia guidelines regarding tests. One thing we did not expect in these procedures is the time delay when communicating with reviewers and project owners. Being an open-source project, it's very possible you are working through your code review with someone on the other side of the world, meaning it could be *hours* before you receive feedback for every modification you make to your pull-request. We recommend starting communication with project owners and possible reviewers very early, even before you initiate a pull-request if possible, in order to counteract any delays in communication.

Writing Oracles for Test Cases:

Much of our task work involved writing test cases. As discussed before, the only criteria for successful test cases described in the task outline was generating coverage. However, Oppia uses unit testing on the backend and we know this means the need for a test Oracle (in this case via assertions). As discussed in class, writing an oracle can be one of the most difficult parts of implementing a test and that was also the case for our tasks here. By looking at the coverage reports, it was easy to determine where the test cases needed to cover and it was even relatively easy to generate that coverage in most cases. However, because Oppia is a large project and we were very unfamiliar with the codebase when we started, it was very difficult to determine what effect the test case has and *should* have and thus very difficult to write an assertion for a complete test case. We recommend allocating more time to analyze the relevant parts of the codebase you will be testing, as this will make it easier to write oracles.

Pair Programming in Task Implementation:

In the "Plan updates" section, we mentioned that we utilized pair programming in implementing task 2. We decided that in order to even the workload and ensure both partners get the most out of the task we would alternate between the "Driver" role and "Observer" role. This had the effect of taking a bit longer to write the tests (as expected with pair programming), but also rose coverage 6% from the work accomplished on this task prior to switching to pair programming. This significant rise in implementation effectiveness (i.e. generated coverage) is a great example of how pair programming

makes for better code. Luckily, because this task involved writing unit tests and each test involved a new section of code, the effect of “adding programmers to a late project makes it later” was not seen here. In our case, utilizing pair programming after one programmer “gets stuck” was very beneficial, however we urge caution in doing this for other projects as it could make the project even later.

11. Advice for Future Students

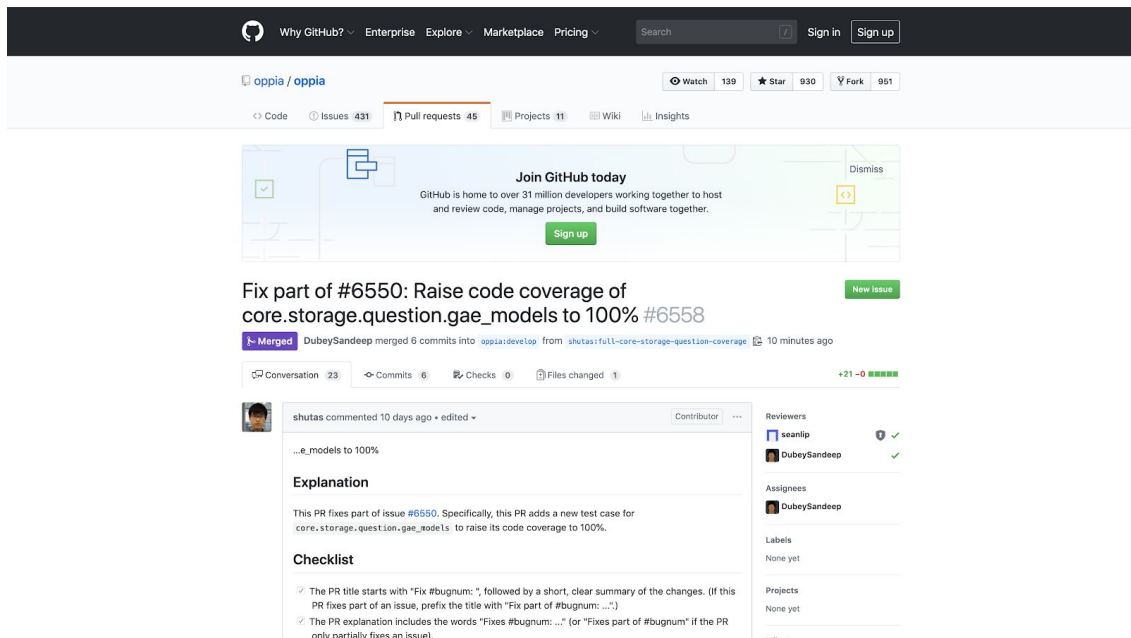
Shuta: "Start assignments early! I hear effort estimation is hard... give enough time for yourself and your programs/tools!!!"

William: “Start HW6 early! Like beginning of the semester early! You’ll get much more out of it and it’s a great talking point for interviews.”

We permit future students to see our material.

12. Optional Extra Credit

Task 1 has been successfully approved by a project owner, and it was merged into the production code.



The screenshot shows a GitHub pull request page for the repository 'oppia / oppia'. The pull request title is 'Fix part of #6550: Raise code coverage of core.storage.question.gae_models to 100% #6558'. It is marked as 'Merged' and was merged by 'DubeySandeep' into the 'oppia:develop' branch from 'shutas:full-core-storage-question-coverage' 10 minutes ago. The pull request includes a comment from 'shutas' stating '...e_models to 100%'. The 'Explanation' section states: 'This PR fixes part of issue #6550. Specifically, this PR adds a new test case for core.storage.question.gae_models to raise its code coverage to 100%.' The 'Checklist' section contains two items, both checked: 'The PR title starts with "Fix #bugnum: ", followed by a short, clear summary of the changes. (If this PR fixes part of an issue, prefix the title with "Fix part of #bugnum: ...")' and 'The PR explanation includes the words "Fixes #bugnum: ..." (or "Fixes part of #bugnum" if the PR only partially fixes an issue).' The right sidebar shows 'Reviewers' (seanlip and DubeySandeep) and 'Assignees' (DubeySandeep).

Link: <https://github.com/oppia/oppia/pull/6558>