6th International Conference on Higher Education Advances (HEAd'20)
Universitat Politècnica de València, València, 2020
DOI: http://dx.doi.org/10.4995/HEAd20.2020.11218

Applying Test-driven Development to Evaluating Student Projects

Cuong Huy Tran, Dragos Truscan, Tanwir Ahmad
Department of Information Technologies, Åbo Akademi University, Finland.

Abstract

Grading software projects submitted by students can become a heavy and time-consuming task, which for many students, can result in delayed feedback provided to them. Additionally, one would like to allow students to evaluate themselves early their projects before submitting the final version for grading.

This paper presents a solution that improves the grading process of student projects not only for lecturers, but also for students. In our approach, we adopt a test-driven development methodology to provide a clear benchmark of the course project implementation. Our approach allows students to self-evaluate their progress at any moment, while lecturers can use it to automate the grading process. GitHub Classroom is used as a supporting tool to allow students to retrieve and implement their projects from the same initial skeleton project including the tests, and lecturers to retrieve the student projects and evaluate them automatically.

The results show that test-driven development is a viable solution to be applied in an academic environment to improve the grading process. This study also shows that courses in Information Technology area could use our approach to increase learning and teaching efficiency.

Keywords: *Test-Driven Development; GitHub Classroom; automated testing; self-evaluation.*

1. Introduction

Grading assignments and course projects have always been an intensive process in higher education. Manual evaluation and grading do not provide the necessary scalability needed for courses with many students. As a concrete example, we refer to one of the courses on the master's level at Åbo Akademi University in Finland, called Development of Web Applications and Web Services (DEWAS). In this course, the students must implement individually a web application, which can contain between 1500 and 2000 lines of code depending on the coding practices and completeness of the implementation.

Evaluating a project in this course is done not only by downloading, building the project, starting the application and navigating through it, but also by inspecting the source code for checking how different features were implemented. On average, grading a project can take around 20 minutes. The course has a variable number of students each year, ranging between 50 and 100, which can result in a high workload for evaluating all projects and providing feedback by the teaching personnel. In addition, we would like that students are able to evaluate their own projects throughout the course, so that they can get an estimate of how many of the project requirements have been implemented and also the corresponding grade for the project.

Our approach applies Test-driven Development (TDD) (Ashbacher, 2003) in building an automatic grading process. TDD is a process in which requirements are turned into test cases, then the code is written to fulfill the tests. This process is usually applied to make sure that every implementation code written meets the requirements. A test case basically executes the system under test (or parts of it) with a sequence of inputs and it checks that the outputs provided by the system corresponds to the output specified by the specification of the system.

Basically, the teachers create a skeleton project with the initial settings and structure, including a set of tests and store it into a software version control system (Chacon & Straub, 2014). The students can download this project to their computers and start working on it. The set of tests should fully reflect the requirements in the assignment specification. Since the project implementation is empty, these tests will fail in the beginning. Students are required to implement the project in order to pass the tests. By continuously evaluating their implementation against the tests, students can track their progress and evaluate their corresponding grade before submitting the final version of the project for grading. After the deadline, lecturers will automatically download and grade all projects at once.

To provide tool support for the above approach, we used GitHub Classroom[1] online version control system, which is one of the tools provided by the GitHub initiative for education. GitHub Classroom provides a means to distribute material, give student feedback, and collect assignments.

2. Related work

Automatic grading of assignments is not a novel topic, many researchers have already addressed this topic in the past with similar approaches. Pilla (2017) utilized GitHub and Travis CI, a continuous integration (CI) service that integrates with GitHub, to build an automatic testing environment for students. Although the work was conducted on some simple C-code assignments, the preliminary results showed great potential. Comparably, Cai and Tsai (2019) applied a similar solution to an Android application development course with improved security.

However, neither of them used a starter repository in their solution. Our approach is also different from theirs because we follow TDD to create a starter repository. Students can download the repository and start working immediately. We do not use any continuous integration (CI) service; instead, we have implemented our approach to automatically download student projects, grade them and generate a detailed course-level report. From our experience, a continuous integration does not provide a global view on all students' repositories, and it requires students to commit code frequently to be relevant. With our approach, we can retrieve student projects any time we want and have all the information of those projects in the report. The approach also allows teachers to update the starter repository and even student repositories.

3. Approach

Figure 1 illustrates the general workflow of our proposed approach. Before the course starts, the teachers specify the project requirements that are directly related to the topics of the course. The system to be implemented has several use cases, for instance, the user should be able to sign up, sign in, sign out, create items, delete items, etc. Use cases can have different levels of complexity and can provide a certain number of points that contribute to the final grade. Each use case is broken into have several functional requirements, for instance the restrictions on the username or the password of the user account. One test will be created for each requirement and added to the test suite.

[1] https://classroom.github.com/

In order to implement the tests before the system is implemented, we need to fix the interface of the system and clearly specify it in a document. During the implementation of the project, the students will implement the behavior behind the specified interface. When the application programming interface (API) is defined and specified and the tests are created, they are both included in a skeleton project which will be delivered to students via GitHub.

GitHub Classroom is an online tool that allows teachers to create assignment repositories. An invitation link is provided to the students. By accessing this link, each student will trigger the automatic creation of a private repository to which both the student and the teacher have access. If a starter code is provided in the initial assignment repository, it will be copied to which of the newly created student repositories. When students download (clone) their assignment repository to the to their computer, they receive a copy of the started code, including the tests, and they can start the implementation of their projects.

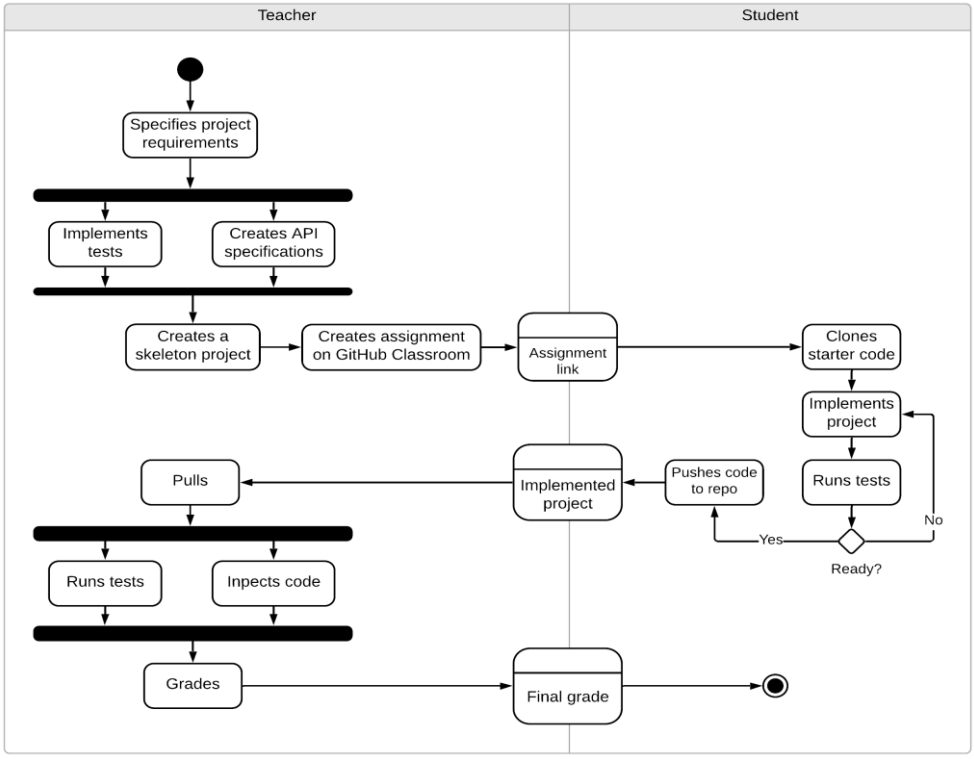


Figure 1. Workflow of the automatic grading process

During the implementation, students have the possibility to run the tests periodically to check what requirements they have implemented successfully and how many points they

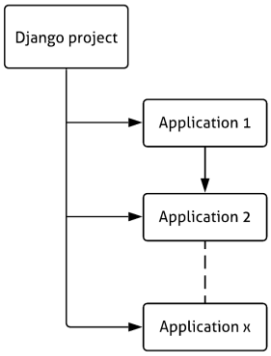
currently have. The tests can be used as a self-evaluation tool to check which use cases have passed and which have failed. When they consider their implementation corresponds to the expected grade or before the deadline, students submit their projects by uploading (pushing) the code to the GitHub Classroom repository.

After the final deadline of the project, teachers will automatically retrieve (pull) all repositories, run all the tests for each of them, and create an overview report including the grades for all the students. Manual inspection of the code can still take place if the teacher considered necessary.

4. Example

For this concrete course, we use the Django framework (Holovaty & Kaplan-Moss, 2009) to develop web applications. Django is based on the Python programming and follows the principles of the *model-view-controller* (MVC) design pattern, which in Django is referred to as *model-view-template* (MVT). This pattern allows the separation of the user interface (template) from the business logic (view) and from the data (model) stored in the system.

The structure of a Django project follows the same pattern. A project can be structured into several applications (Figure 2-left) each with its own folder. There are several mandatory files inside a project folder, however we will only discuss the relevant ones for this paper.




```
myproject/
    myapp/
    myproject/
        __init__.py
        settings.py
        urls.py
        wsgi.py
    db.sqlite3
    manage.py
```

```
# myproject/
myapp/
    static/
    templates/
    __init__.py
    admin.py
    apps.py
    models.py
    urls.py
    tests.py
    views.py
```

Figure 2. Structure of a Django project.

At the top level of the project (Figure 2-middle), the url.py file specifies the interface of the web application as a list of links and what functionality of the application to invoke in order to provide a response to the client (browser). For example, in the code below, whenever a browser will send an HTTP request to an address *https:/myserver.com/signup/* the Django framework will map the request to a function called *signout_view()* which implements the business logic of the sign out feature of the application.

```
urlpatterns =[
   ….
   path('signout/', user.views.signout, name='signup_view'),
]
```

The implementation of the *signup_view()* function is located in one of the *views.py* files of the Django project folders (Figure 2-right). This function, similarly, to all the other business logic functions in the project, have an empty body in the project skeleton which should be completed by the students. After being implemented, the function will return a HTML page that will be sent back to the browser to be displayed.

A corresponding test for this function is shown below. The test verifies requirements REQ1.1 (lines 2-3) by sending a HTTP POST request to the *signup/* URL (line 9) and providing a set of parameters via the *context* variable defined at lines 4-8. The test expects (line 10) that the application will return a HTTP response message with status code 302, in which case the test will be marked as PASS otherwise as FAIL. When the test is successful (PASS), line 12 will be executed and the number of points scored by the entire project will be increased.

```
1       def test_sign_up_with_valid_data(self):
2         #REQ1.1 Sign up with valid username, password and password confirmation,
3         # should return status code 302
4             context = {
5              "username": "testUser3",
6              "password": "!@ComplicatedPassword123",
7              "email": "user1@mail.com"
8             }
9             response = self.client.post("signup/", context)
10            self.assertEqual(response.status_code, 302)
11            # calculate points
12            self.__class__.number_of_passed_tests += 1
```

With this approach, students can check the tests frequently to check their progression. After each run, a report will show what tests have failed or passed and how many points a student has currently earned. Students can go to each test case and test method to inspect the test failure in more detail. Figures 3 and Figure 4 show examples of failed and respectively passed test when using PyCharm IDE[2], a recommended IDE in the course. After each run the students can visualize the number of points received by the project.

[2] https://www.jetbrains.com/pycharm/

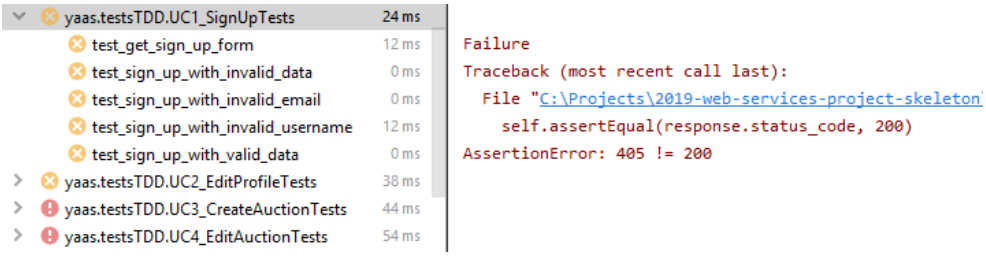


Figure 3. Example of failed tests.

As mentioned in the previous section, we have implemented a script in Python to support the grading process by the lecturers. When the project deadline has passed, the teachers will execute the script, which will run the provided tests on each submitted project and provide statistics with the use cases passed and how many points have been received by each project. A generic example is shown in Table 1.

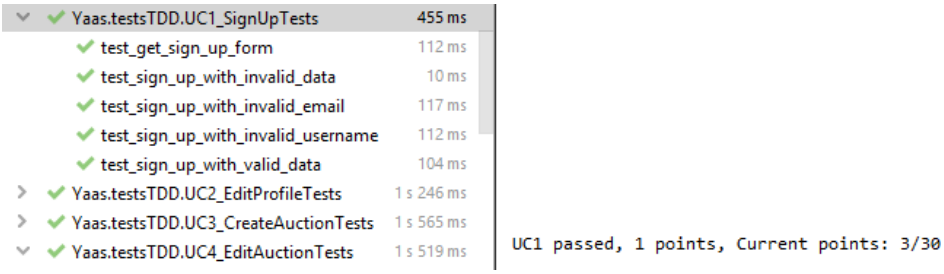


Figure 4. Example of passed tests.

In total, we have implemented 41 tests, covering 12 case studies. Having followed the TDD approach, where the tests have been provided before the project was implemented allowed the students to self-evaluate themselves during the implementation with respect to how many points the project is worth, which functions introduce errors, and what code needs to be improved. If the students commit their code regularly to repository before the deadline, teachers could check their progress status and give helpful feedback during the course rather than only acknowledge the final product.

Table 1. Example of the grading report.

Student	Date	UC1	UC2	…	Total	Repo link
Student A	16/01/2020	1	1	…	16	https://github.com/…
Student B	16/01/2020	1	0	…	18	https://github.com/…
Student C	16/01/2020	0	1	…	15	https://github.com/…

5. Evaluation

To this extent, most of the students provided positive feedback on the approach since it allowed them to understand and plan their programming tasks better.

The approach also has some limitations. In order to provide the automated tests from the beginning, we had to clearly specify the interface of the application, for which otherwise the students would have had complete freedom. Also, some of the features of the application could be implemented in different ways by the students, but the tests would impose that a certain way is followed, creating additional constraints.

Another negative aspect was that not all requirements could be translated into automated tests. For those we had the option of either removing them from the project requirements or manually inspect them when the project is submitted.

Some effort was spent in the beginning by the lecturers to implement the tests and clearly specify in the interface, however the benefits in terms of efficiency the new approach decreased dramatically the time needed to grade the assignments. As such, we were able to run the automated tests on all 60 projects submitted by students in around 110 minutes on a Windows 10 laptop featuring an Intel i7-7500U CPU with two cores at 2.90GHz and 16GB of RAM. This menas less than two minutes per project. Roughly 5 minutes of additional time was allocated in average to manual code inspection of the requirements that had no associated tests. Overall, we have observed a reduction of time of more than 65%.

6. Conclusions

In this paper, we presented an approach that uses test-driven development for automatic grading and evaluation of student projects in a programming course. The results show that TDD is a viable approach which in combination with automation tools provided by for instance GitHub Education can make the learning process more efficient. As future work we plan to evaluate our approach in other courses and also calculate its impact on the average grade of the students in the course.

Acknowledements

This work has received partial funding from the Electronic Component Systems for European Leadership Joint Undertaking under grant agreement No 737494. This Joint Undertaking receives support from the European Union's Horizon 2020 research and innovation programme and Sweden, France, Spain, Italy, Finland, the Czech Republic.

References

Ashbacher, C. (2003). Test-Driven Development: By Example, by Kent Beck. *The Journal of Object Technology*, *2*(2), 203. doi: 10.5381/jot.2003.2.2.r1.

Cai, Y.-Z., & Tsai, M.-H. (2019). Improving Programming Education Quality with Automatic Grading System. *Lecture Notes in Computer Science Innovative Technologies and Learning*, 207–215. doi: 10.1007/978-3-030-35343-8_22.

Chacon, S., & Straub, B. (2014). *Pro Git* (2nd ed.). Berkeley, CA: Apress.

Fowler, M., & Foemmel, M. (2006). Continuous integration. *Thought-Works) http://www. thoughtworks. com/Continuous Integration. pdf*, *122*, 14.

Holovaty, A., & Kaplan-Moss, J. (2009). *The definitive guide to Django: Web development done right*. Apress.

Pilla, M. L. (2017). Teaching Computer Architectures through Automatically Corrected Projects: Preliminary Results.