Applying Test-Driven Development for Improved Feedback and Automation of Grading in Academic Courses on Software Development

Truscan, Dragos; Ahmad, Tanwir; Tran, Cuong

Published in:
Frontiers in Software Engineering Education - 1st International Workshop, FISEE 2019, Invited Papers

DOI:
10.1007/978-3-030-57663-9_20

Published: 01/01/2020

Document Version
Accepted author manuscript

Document License
Publisher rights policy

Applying test-driven development for improved feedback and automation of grading in academic courses on software development

Dragos Truscan[0000-0002-4367-6225], Tanwir Ahmad[0000-0003-3416-2422], and Cuong Huy Tran[0000-0003-1127-4659]

Faculty of Science and Engineering, Åbo Akademi University, Finland
firstname.lastname@abo.fi

Abstract. Grading student assignments and projects in software development courses is a time-consuming task. The lecturer has to download individually each assignment, compile it and manually check that the implementation satisfies the requirements. In addition, the students would like to get early feedback on their solutions, not only as guidelines on whether their solution meets the expectations of the lecturers, but also a way to estimate the current number of points their solution deserves. In this work, we propose the use of the test-driven development process as an approach to both guide the students during the implementation of their projects and as a way to speed up and make the grading process more scalable. Furthermore, we show how we take advantage of community-based software development tools such as GitHub to support our approach. We evaluate the proposed approach by applying it to an academic course for developing web applications. The results show that the approach reduces the grading effort by 60% and that the early feedback it provides was appreciated by students.

Keywords: Test-driven development. Test automation. Academic course. Software development. Course self-evaluation.

1 Introduction

Evaluating student projects in academic courses on software development can be a tedious and time-consuming task. In such projects, a software application is typically developed either individually or in groups by students. Lecturers formulate the requirement of the application and then students develop it before the deadline of the task. Then the students submit their project for grading, typically by uploading the project files to a course management system such as Moodle. After the deadline, the lecturers download the project, execute it, and check that the application requirements are satisfied. Then, the lecturers provide feedback for the solution and a grade for the project.

There are two issues with the above approach. First, the students receive feedback and a grade for their project only after the submission deadline and evaluation period needed by the lecturers. Receiving earlier feedback, during the development of the project, would allow students to evaluate better their work and efforts needed to complete the project. The second issue is related to the time needed by the lecturers to check the project of all the students in the course. For a large number of student projects, it may take several days or weeks before all the submissions are evaluated.

As a concrete example, in a course on developing Web applications at our university, the size of a completed project is between 1500 and 2000 lines of code. On average, grading a project takes 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.

Based on previous experience, following an incremental software development approach for the projects would be beneficial for students in receiving feedback faster, but will increase the amount of work of the lecturers compared to checking the project at the end of the course. This is because the features implemented in past versions have to be rechecked in case they may have been updated. So for every increment of the project more time has to be allocated per student and, in the end, in the last increment the complete project will have to be checked anyway.

Test-driven Development (TDD) is a software development process that promotes the development of software based on short iteration cycles. The starting point is a set of tests that are created, typically from the requirements of the system, before the implementation of the system is available. During each cycle, one or several features of the product are implemented to make one or several of the tests pass. When all the provided tests pass, the development of the software is considered complete.

The proposed approach applies TDD for evaluation and grading of student projects. We create a set of acceptance tests that are provided to students at the beginning of the project. These tests are used as a reference by both the students during the implementation of their projects and by lecturers to evaluate the solutions implemented by students. The approach allows the students to receive continuous feedback during their work on the quality of their solutions and simplifies the grading process by the lecturers. To automate our approach, we use the Github repository hosting service and a set of custom scripts. Using our approach, the lecturers can save time from the grading process and allocate it to providing more in-class feedback during the course.

The work presented in this chapter is an extension of the work published in (Cuong Huy Tran, Dragos Truscan, Tanwir Ahmad 2020). We extend the previous work with a more thorough introduction of the software development concepts and more details on the approach. Moreover, we provide more details on the case study and its evaluation.

The structure of this chapter is as follows. We start by introducing different concepts of the software engineering field that are relevant for this chapter. Section 3 introduces a generic approach in which TDD is employed for grading student projects and discusses the design decisions and the benefits of the approach. Section 4 presents a case study on how we have applied the approach in practice. We analyze the results in Section 5. Finally, we draw conclusions in Section 6.

2 Software development concepts

Traditionally software is developed in phases starting from the requirements of the application, then its design and implementation. When the implementation is completed, it is tested to see if it satisfies the requirements. In software testing, the implementation (code) is executed with different *test inputs* and the *test outputs* are checked if they correspond to the *expected outputs*. The latter are typically derived from the requirements or specification of the software. Whenever the test outputs correspond to the expected outputs we assign a *passed* verdict to the test, otherwise a *failed* one.

The development phases are typically combined into different software development processes such as waterfall, agile, etc depending on the characteristics of the application to be developed and of the structure of the development team.

2.1 Test-Driven Development

Originating from Extreme Programming practices, Test-Driven Development (TDD) (Beck 2003) is a software development process that requires tests to be written before the implementation of the code is started. The TDD process is a cycle that is repeated over and over until all the tests pass (Beck 2003), as shown in Figure 1.

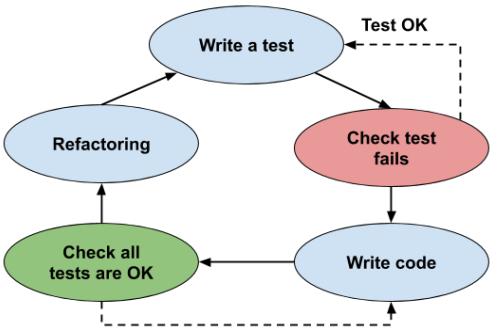

Fig. 1. Test-driven development cycle.

- **Write a test.** Every new feature begins with writing a test. The test should be brief and clearly expressed. Writing a test before the code is implemented motivates the developers to think first about the requirements, the design of the system and the way it should work.
- **Run and check if the test fails.** The test is expected to fail since the application code does not exist yet. This step emphasizes the target feature for the developers. If the test passes, it must be re-written to fail.
- **Write code.** Write just enough production code to fulfill the test. Programmers need to be careful not to implement further than the functionality of the test.
- **Run all tests.** If all tests pass, it means the new code does not break any existing features and the new test is satisfied. If they fail, the new code has to be modified until all tests pass.
- **Refactor code.** In this step, the code is refactored, by cleaning it up, removing duplication, or improving its readability and maintainability. The test cases are re-run frequently to ensure the refactoring code does not alter unrelated features.

The main benefit of TDD is that when writing new code, the test cases can act as a guideline, so the developers can conveniently follow, resting assured that they are on track and no feature's specification is missing (IBM n.d.). Additionally, by running tests throughout the development process, feedback is given regularly and no code left untested. Moreover, developers spend less time on debugging and fixing errors. Although TDD is not a miracle solution to eliminate all bugs, more tests mean better code coverage, and that will reduce the cost of maintenance and a large number of bugs (IBM n.d.). Combined with a version control system, when a test fails, TDD helps to identify the error quickly and more productively. TDD can also lead to more clean, modularized, and extensible code because of the constant refactoring. The code is tidier, well documented, which allows other team members to understand it. This makes the application under development more suitable for future enhancement or expansion.

TDD also has some limitations. Different authors report that TDD's slow learning curve makes it difficult to adopt. In addition, the final product may be too biased by the way the tests were created and the requirements provided may not be complete or well-specified. Furthermore, if the project specifications and requirements are not studied and analyzed well enough, passing tests could cause a false sense of safety. Due to the nature of TDD, it has a long learning curve. Additionally, writing and maintaining an overwhelming number of tests costs time and resources, particularly for small teams. It takes approximately as much as 16% more development time than that of the traditional approach where tests are created after the implementation is completed (George and Williams 2004).

2.2 Software version control system

A version control system (VCS) (Spinellis 2005) is a tool that helps developers to manage changes to source code over time so that they can recall them later if needed. VCS keeps track of every modification from add or edit to move or delete in a special kind of database. The types of file VCS can track are not only source code, but also images, audio files, movies, or any other type of digital asset.

For almost all software projects, the source code is the most critical central part, and the teams are responsible for protecting it. A VCS, which is updated frequently during the development, can also act as a backup storage. If some files are lost due to accidents or human error, the team can quickly recover them from VCS.

There are two popular types of version control systems: centralized and distributed. Centralized version control systems store all files and the full version history in one shared server. The developers retrieve some of the source files from the central location, modify it and store it back to the central location. In contrast, in distributed VCSs, the developers completely mirror the project or repository, including the full version history. Then they make changes locally to the files and submit them later to the centralized location.

Git is one of the most popular open-source versioning control systems and several deployment servers are available for public use. For instance, *github.com* is a Git repository hosting service where developers can version and share their software. It provides services for both public and private repositories. It offers several additional functionalities, such issue tracking system, wiki pages, etc.

One interesting feature of github.com is that it is free to use for educational purposes via the GitHub Classroom initiative. GitHub Classroom allows lecturers to create assignments for which students submit code via the VCS, track student progress, and integrate with useful third-party tools. It also scales up for courses with a large number of students.

3 Approach

The proposed solution is to apply the concepts of TDD to evaluating and grading student projects. We provide students with a set of acceptance tests at the beginning of the project to be used as a reference by both the students during the development of the project and by lecturers to evaluate and grade the project after the deadline. The students are not allowed to modify the provided acceptance tests, but they can add additional tests if they consider them helpful for their implementation.

The approach is illustrated in Figure 2. The requirements of the project are first specified. Then, the lecturers implement a reference project (similar to the one expected to be delivered by students). The set of tests is created from the

requirements of the project by the lecturers. However, in order to execute the tests against the implementations created by the students, lecturers need to decide and clearly specify the interface of the application in advance. The tests are executed to verify the implementation of the project. This is an iterative process which ends when tests for all requirements have been implemented.

The requirements specification, interface specification and the tests are used to create a GitHub Classroom assignment. The assignment link is provided to students. Whenever a student accesses the link, a new source code repository is automatically created on GitHub, to which both the student and the lecturers of the course have access. If a starter code is provided in the initial assignment repository, it will be copied to the newly created student repository. When students download (clone) their assignment repository to their computer, they receive a copy of the started code, including the tests, and they can start the implementation of their projects.

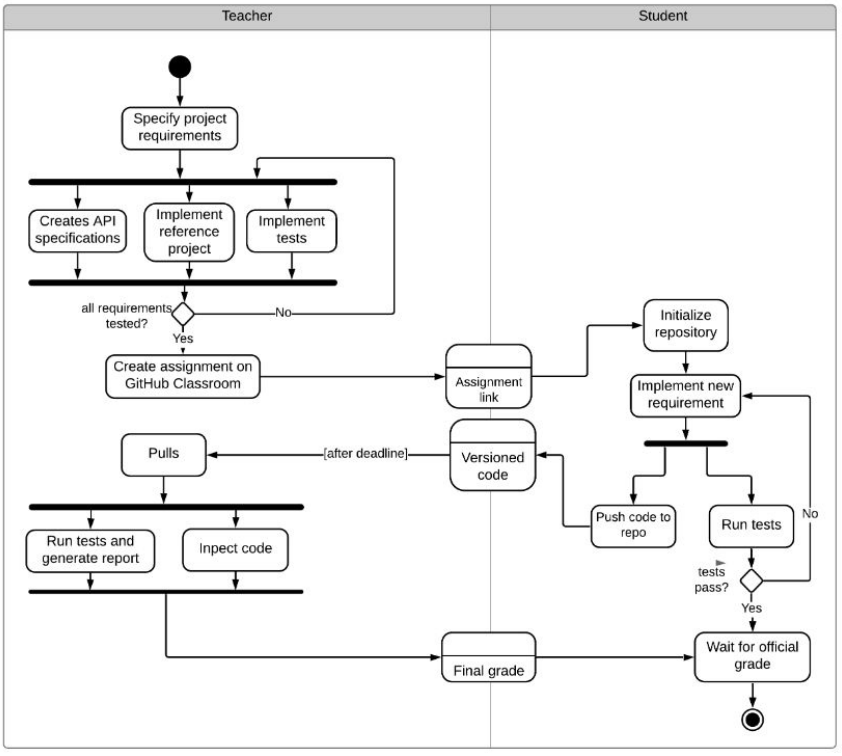

Fig. 2. Workflow of the proposed approach

The first time the tests are run against the project they will all fail since the project is not yet implemented. The students will proceed with developing their project and can run the tests regularly. As more features are implemented, the tests will start passing. The tests serve as self-evaluation to the students on the

progress of their project. At the same time, the students should push their project regularly to the repository for versioning and backing up the code.

When the deadline of the project is over, the code is already available in the repository and the lecturers can evaluate the projects by pulling all student projects from their corresponding repositories and running the tests to check the progress and the completeness of the projects. In our approach, the last two activities are performed automatically using a set of scripts and the Application Programming Interface (API) of GitHub.

When the deadline of the project is over, the code is already available in the repository. Lecturers can pull students' projects and run acceptance tests to evaluate their progress and completeness. In our approach, pulling and running tests are performed automatically using a set of scripts and the Application Programming Interface (API) of GitHub. Based on the result of the tests, lecturers can create an overview report including the grades. Manual inspection of the code can still take place if lecturers consider it necessary.

4 Case study

As an example of our approach, we show how we have applied it in practice in an academic course on the development of web applications. In this course, the students have to develop a web application, called YaaS (Yet Another Application) similar to ebay.com, in which different users (sellers) can create auctions to sell products, whereas other users (buyers) can make bids on ing auctions. When the deadline for a given bid passes, the auction is adjudicated to the highest bidder and the seller, buyer and other bidders are notified.

A Web Application is a computer program that provides dynamically created content to be displayed in a web browser (Shklar and Rosen 2003). The information between the client (i.e., the web browser) and the server is exchanged via the HyperText Transfer Protocol (HTTP) (Fielding and Reschke 2014). HTTP is a stateless request/response protocol. In a typical interaction, the client submits a request to a server, the server processes the request and returns a response to the client, most often formatted using the HyperText Markup Language (HTML) (Krause 2016).

In this course, we required that the YaaS application was implemented by students using the Django Web framework (Holovaty and Kaplan-Moss 2009), which is based on the Python programming language (van Rossum et al. 2008).

4.1 Initial Project artefacts

In order to apply the process proposed in Section 3, the lecturers of the course created three artefacts to be delivered to students: requirements specification, interface specification, and the acceptance tests. We detail them in the following.

Requirements specification document. The requirements of the project are specified using use cases, for instance, the user should be able to sign up, sign in, sign out, create items, delete items, etc. In total, the YaaS application has 12 use cases, each having different levels of complexity. Each use case is decomposed into several functional requirements, such as the user should be able to log in with valid credentials or an error message should be displayed if invalid credentials are used. To summarize, the YaaS application has 41 functional requirements.

With respect to the grading of the project, each use case gives a predefined number of points depending on its complexity. The number of points given by each use case is clearly mentioned in the requirements specification document as a hint to the students on the importance and expected complexity of the use case. The points for a use case are obtained only if all the requirements associated with the use case are implemented correctly. This grading approach is specific for this particular course and project, but it can be customized in other courses.

Interface specification. An interface specification is created to reflect all the requirements of every use case in terms of the interface of the application. The interface specification file describes what are the URLs used by each user case, what HTTP requests can be sent to those URLs, what parameters they require, and what is the expected response. An example of interface specification for use case UC1-Create user account is given in the following Table 1.

Table 1. Example of interface specification for UC1

Use case	UC1 - Create a user account
URI	/signup/
Allowed HTTP methods	GET – get a signup form, return code 200. POST – create a user with username and password ● Sign up without a password, means invalid data, return status code 200. ● Sign up with an already taken username, return status code 400, and an error message is present in the response content (HTML). ● Sign up with an already taken email, return status code 400, and an error message is present in the response content (HTML). ● Sign up with valid data, return status code 302 because the page would redirect to the index page after a successful signup.
Example request:	HTTP1.1 POST /signup

Example expected response:	HTTP1.1 302 Redirect
	{
	“username”: “user1”,
	“password”: “Password1”,
	“password1”: “Password1”,
	“password2”: “Password1”,
	“email”: “user1@mail.com”
	}

Acceptance tests. In order to verify that different project requirements are implemented successfully by students, we create one or several tests for each requirement. Since each requirement belongs to one use case, we group the tests belonging to requirements of the same use case under one test case (see Figure 6). The tests are implemented based on the given interface specification. For convenience we have implemented them in the Python programming language, using the Python unit testing library. However, other programming languages can be considered because the application interface is clearly specified and the acceptance tests are not dependent on the programming language used for the implementation of the application.

For instance, use case UC1 has five requirements. One of the tests for one of the requirements is shown in Figure 3 as a test method. The test verifies requirements REQ1.1 (lines 2-3) by sending an 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 an HTTP response message with status code 302, in which case the test will be marked as PASS otherwise as FAIL.

```
1    def test_sign_up_with_valid_data(self):
2    # REQ1.1 Sign up with valid username, password and
3    password confirmation, 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
```

Fig. 3. Example of a test of requirement REQ1.1

When the test is successful (PASS verdict), line 12 will be executed and the number of points scored by the entire project will be increased by 1.

4.2 Support for automatic grading

Every test case corresponding to a use case has some class-level variables to track and show the number of tests, passed tests, and points of the test case, as shown in Figure 4.

```
1    number_of_passed_tests = 0 # passed tests in this test case
2    tests_amount = 5 # number of tests in this suit
3    points = 1 # points granted by this use case if all test pass
```

Fig. 4. Example of points awarded for a given use case

When a test case completes its execution, a global method is invoked to calculate points aggregated from the individual tests. The method in Figure 5 checks if all tests of the test case are passed (line 3). If there is a failed test, the system will prompt a failure message (line 4). Otherwise, the method adds the points of this use case to the total number of points of the project (line 6-7) and the system will print a success message (line 11) to the user.

```
1     def calculate_points(number_of_passed_tests, amount_of_tests,
2                          points_of_the_use_case, use_case_name):
3         if number_of_passed_tests < amount_of_tests:
4             print("{} fails!".format(use_case_name))
5         else:
6             global current_points
7             current_points += points_of_the_use_case
8             msg = """{} passed, {} points,
9                   Current points: {}/30""".format(use_case_name,
10                  points_of_the_use_case, current_points)
11            print(msg)
```

Fig. 5. Code for calculating the points of the project

4.3 Feedback to students

During the course, the students receive three types of feedback:

- From the execution of the acceptance tests, students receive feedback when a feature is implemented or not (if its tests pass or not). In addition, we have tried to implement the tests to provide informative error messages. As mentioned in the paper, after each execution of the tests, the students get an automated evaluation of the grade of the project. This is a continuous process.
- Throughout the course, during lectures and labs, the students can ask questions on different aspects related to the teaching material, coding practice or the project implementation from course assistants and lecturers. This is also a continuous process.
- When their project is evaluated, besides checking the project with automated tests, the lecturers also inspect the code and provide the final feedback on the project.

In the following, we will focus on the first type of feedback that is an outcome of our proposed method.

At the beginning of the project implementation, all acceptance tests will fail, since no implementation is yet available. A simplified example of a test report where all the tests fail is shown in Figure 6.

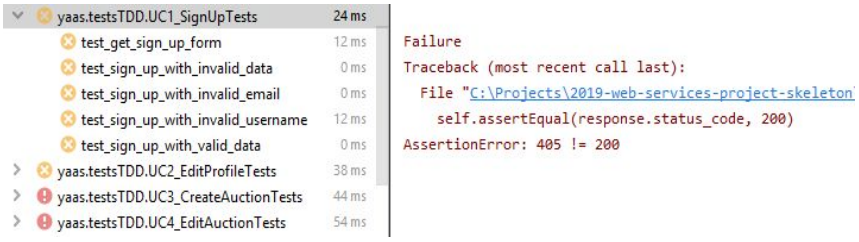

Fig. 6. Example of failed tests.

By having the acceptance tests readily available, the students can check at any moment the status of their implementation. After each execution of the acceptance tests, a report will show what tests have failed or passed and how many points a project has currently earned. Students can inspect the test failure in more detail. Figure 7 shows the test report for the same tests as in Figure 6, when the functionality of the web application satisfies the requirements of the project.

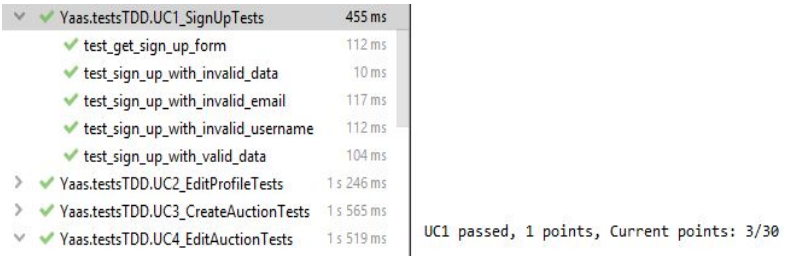

Fig. 7. Example of passed tests.

The students should frequently commit their projects to the GitHub for backup and versioning purposes. When the deadline for project submission has passed, the latest version in the repository will be considered for grading.

4.4 Support for automatic grading by lecturers

In order to automate the grading process, a set of scripts has been implemented to automate different steps performed by the lecturers. The scripts, written in Python, use the GitHub API to download all student projects from GitHub Classroom and store them in a local folder. Then, they execute the tests on each project and save the test report results in a grading report file with the structure presented in Table 2. For each student, the report includes: name of the student, date of running the script, points received by each use case, the total number of points earned by the student, and the link to the repository of the project.

Table 2. Example of the grading report for the course

Student	Date	UC1	UC2	…	Total	Repo link
Student A	25/03/2020	1	1	…	16	https://github.com/…
Student B	25/03/2020	1	0	…	18	https://github.com/…
Student C	25/03/2020	0	1	…	15	https://github.com/…

These scripts can be run not only at the end of the course after the deadline for project submission has passed, but also regularly (e.g., weekly) to check the progress of the students during the course. This allows them to provide additional support or change the pace of the lectures according to the needs of the students.

5 Discussion and evaluation

As discussed earlier in this work, TDD brings some benefits but it may also have some limitations. In order to cope with the slow learning curve, we have provided detailed requirements and interface specifications, and a project skeleton to facilitate quick adoption of TDD concepts. In order to make sure that the requirements and the tests were well-specified, the initial effort was allocated by lecturers to create the tests, the reference project, and the interface specification. Having the reference project implemented in advance, also allowed us to make sure that all requirements are testable and to detect and remove possible inconsistencies.

Another perceived limitation of TDD, is that one can create an implementation that passes the tests without implementing the expected behavior of the application and thus providing a false level of confidence. In our approach, this risk is reduced by the way the tests were designed. Some tests were inherently dependent on each other and sharing data. For instance, one test checked if the user can create an account, another test checked if the user can log in with the specified account which should have been created by the previous tests. This is not a complete bullet-proof approach, and for that reason, the lecturers also inspect the code manually to detect possible problems practices.

Additional effort has been required to specify the application interface, but this was a tradeoff for having automated tests for the project. When creating the interface several design decisions had to be made which limited the implementation freedom of the students, in our opinion, but that was an acceptable compromise and we consider that it still satisfied the learning objectives of the course.

For the YaaS application, we have implemented 41 tests in total. We have evaluated the approach in one edition of the course in which 60 students submitted projects. After the deadline, 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 means less than two minutes per project. Roughly 5 minutes of additional time was allocated on average for manual code inspection. This activity was largely performed for giving feedback and recommendations to the students. Overall, we have observed a reduction of more than 65% in the grading time.

The submitted student projects, which received the highest grade, had more than 90% of the project requirements implemented and between 1490 and 2050 lines of code. The acceptance tests we provided achieve between 77% and 91% coverage of the source code, which shows that the acceptance tests give a good metric for the overall quality of the project.

The feedback from the students, collected via interviews and course feedback forms, was in general positive. Most of them liked the approach and considered useful to have the acceptance tests available from the beginning. In addition, they appreciated not only the fact that they could estimate the grade in advance, but they can also utilize the tests as guidelines during the development of their project. However, there were some students that considered that the TDD approach and the use of GitHub for versioning required a different mindset and new technical skills. Nonetheless, we consider that these technical skills are useful and mandatory for any computer engineering student.

Based on this preliminary evaluation, we plan to re-apply the approach in the next editions of the course and, in addition, to extend it to other software development courses at our university.

6 Related work

Automatic grading of assignments is not a novel topic and several researchers have already addressed this topic in the past with similar approaches.

Edwards (S. H. Edwards 2003) presents his vision and tool support for automatic grading (S. Edwards 2003) in which TDD should be used in all programming assignments starting from the first year of the Computer Science education. Differently from our approach, Edwards suggests that the students are required to create their own tests to accompany the code that they write, and these tests are evaluated against a reference implementation. Similar to our approach, he proposes an automated assessment tool to which the students submit their code, with the difference the tool is assessing both the correctness of the student tests and of the application. In addition, the tool provides static checks and feedback on the coding style which in our approach is performed manually in class and at the end of the course.

Janzen and Saiedian (Janzen and Saiedian 2006) propose *test-driven learning* as a way of using TDD for teaching both testing and programming. In practice, they suggest that different programming examples and small

assignments are accompanied by tests (*assert* statements) that would indicate to students both the expected interface and the expected behavior of the program. The main benefits perceived from their approach is that they can improve the teaching of programming via examples accompanied by tests. Differently from their approach, the goal of our work is automated acceptance testing of student programming projects as a way of guiding the students during their work.

Pilla (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 (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 should 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 student repositories, and it requires students to commit code frequently to be relevant. With our approach, we can retrieve student projects at any time we want and have all the information about those projects in the report. Our approach also allows lecturers to update the starter repository and even student repositories.

7 Conclusions

This chapter introduced an automated approach for evaluating student projects by employing the concepts of the test-driven development approach and by taking advantage of community-based tools such as GitHub. We have applied and evaluated the approach in an academic course on developing web applications. Even though we used a specific development framework in that course such as Django, the approach can be easily adapted and applied with other tools and development environments.

The approach required some extra efforts in the beginning, when creating the tests and the interface specification and developing the scripts used for automatic grading and reporting of all student projects. But these artefacts were created only once and they can be reused in future editions of the course.

Depending on the course settings, the implementation of the reference project can be omitted, which will be in the true spirit of TDD. However, to make sure that the expectations from student projects are realistic we consider it advisable.

The evaluation showed benefits with respect to both the early feedback that the approach provides to students, but also in speeding up the course grading process. The latter makes the approach a good candidate for online courses with a large number of participants.

In future work, we plan to evaluate the approach in future editions of the course and to measure its impact on the grades of the students. To this extent, we plan to run controlled experiments in which a part of the students will use the TDD approach and the other part the manual approach. In addition, we plan to reapply the approach in other courses on software development in order to evaluate its benefits and limitations.

Last but not least, we consider that by having an automatic grading approach, we are not aiming at minimizing the lecturer-student interaction, but by providing clear quantifiable expectations on the course goals and in automating tedious tasks.

Acknowledgements

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

Beck, Kent. 2003. *Test-Driven Development: By Example,*. Boston, MA: Addison-Wesley.

Cai, Yun-Zhan, and Meng-Hsun Tsai. 2019. "Improving Programming Education Quality with Automatic Grading System." In *Innovative Technologies and Learning*, edited by L. Rønningsbakk, T. T. Wu, F. Sandnes, and Y. M. Huang, 11937:207–15. Lecture Notes in Computer Science. Springer, Cham.

Cuong Huy Tran, Dragos Truscan, Tanwir Ahmad. 2020. "Applying Test-Driven Development to Evaluating Student Projects." In *6th International Conference on Higher Education Advances (HEAd'20)*.

Edwards, Stephen. 2003. "Using Test-Driven Development in the Classroom: Providing Students with Automatic, Concrete Feedback on Performance." In *Proceedings of the International Conference on Education and Information Systems: Technologies and Applications EISTA*. Vol. 3. http://web-cat.org/publications/Edwards-EISTA03.pdf.

Edwards, Stephen H. 2003. "Rethinking Computer Science Education from a Test-First Perspective." *Companion of the 18th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications - OOPSLA '03*. https://doi.org/10.1145/949344.949390.

Fielding, R., and J. Reschke. 2014. "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing. RFC 7230."

George, Boby, and Laurie Williams. 2004. "A Structured Experiment of Test-Driven

Development.” *Information and Software Technology* 46 (5): 337–42.
Holovaty, Adrian, and Jacob Kaplan-Moss. 2009. “The Definitive Guide to Django.” https://doi.org/10.1007/978-1-4302-1937-8.
IBM. n.d. “Test-Driven Development.” Accessed April 20, 2020. https://ibm.com/garage/method/practices/code/practice_test_driven_development/.
Janzen, David S., and Hossein Saiedian. 2006. “Test-Driven Learning.” *Proceedings of the 37th SIGCSE Technical Symposium on Computer Science Education - SIGCSE ’06*. https://doi.org/10.1145/1121341.1121419.
Krause, Jörg. 2016. “HTML: Hypertext Markup Language.” In *Introducing Web Development*, 39–63. Apress, Berkeley, CA.
Pilla, Mauricio Lima. 2017. “Teaching Computer Architectures through Automatically Corrected Projects: Preliminary Results.” *International Journal of Computer Architecture Education* 6 (1): 62–67.
Rossum, Guido van, Raymond Hettinger, Nicholas Coghlan, Jack Diedrich, David Beazley, and David Mertz. 2008. *The Python Programming Language*. Prentice Hall Open Source Software Development Series. Prentice Hall PTR.
Shklar, Leon, and Richard Rosen. 2003. *Web Application Architecture: Principles, Protocols and Practices*. Wiley.
Spinellis, D. 2005. “Version Control Systems.” *IEEE Software* 22 (5): 108–9.