# Software Testing as Medium for Peer Feedback

Léon McGregor
Manuel Maarek
lm356@hw.ac.uk
M.Maarek@hw.ac.uk
Heriot-Watt University
Edinburgh, Scotland, UK

## ABSTRACT

Peer feedback has been shown to benefit students' learning as it develops their critical thinking and provides more immediate feedback. It relies on the analysis by students of their peers' work following structured criteria. We propose to apply the same principle to programming courses but in doing so by using techniques of software evaluation: software testing. Students run tests on their peers' code and engage in peer feedback discussions on the results of the test runs. In this paper we present a testing-based code peer feedback learning activity, we describe its technical framework and integration with GitLab, and we discuss its deployment in an undergraduate course and initial evaluation.

## CCS CONCEPTS

• **Social and professional topics** → **Computer science education**; **Software engineering education**; • **Applied computing** → **Collaborative learning**; • **Software and its engineering** → **Software testing and debugging**.

## KEYWORDS

peer feedback, software testing, programming education, testing education, code review education

**ACM Reference Format:**
Léon McGregor and Manuel Maarek. 2020. Software Testing as Medium for Peer Feedback. In *United Kingdom & Ireland Computing Education Research conference. (UKICER '20), September 3–4, 2020, Glasgow, United Kingdom.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3416465.3416474

## 1 INTRODUCTION

Practice is essential for the learning of programming. Problem based learning and learning by failure have proved effective in computer science education [7, 13, 19]. At undergraduate level, students are often expected to undertake practical programming coursework exercises. The students receive feedback on their implementation from the teacher in-class and on their final submission. After the submission deadline, the teacher will assess each solution and provide individual feedback. This process has several challenges.


*UKICER '20, September 3–4, 2020, Glasgow, United Kingdom*

ACM ISBN 978-1-4503-8849-8/20/09…$15.00
https://doi.org/10.1145/3416465.3416474

*Late Feedback.* The feedback associated with a summative assessment takes longer to prepare by the course teacher as it requires a higher quality and more detail than other forms of formative feedback. This results in such feedback reaching the students too late for positively impacting their learning as they often would have already moved on to other courses and other tasks. Meanwhile, the immediate aftermath of a submission is the moment when the problem is fresh in the mind of students.

*Limited Self-Evaluation.* Coursework tasks have a focus placed on producing a solution, with little interest from the students in analysing and evaluating the work they produce. This common lack of interest or limited practice in critical code evaluation is combined with software testing often being neglected in computing programmes [2, 8, 22].

*Isolated learning.* When coursework exercises are individual, which is often a necessity to properly ensure that each student meets the learning objectives of the course, the students work in isolation and could lack peer interactions. Others are wary of plagiarism when interacting with their peers.

### 1.1 Motivation and Inquiry

To address these challenges, we propose to combine and adapt peer feedback and software testing into a learning activity that we named *Peer-Testing* which we incorporate into a coursework exercise process to improve students' learning and learning experience.

*Peer feedback* or *peer assessment* are processes that invite students in a cohort to review each other's submissions. Peer assessment is more effective when the evaluation criteria is correctly explained, and when students assess multiple peers, instead of focusing on one-to-one peer interactions [6]. Peer assessment or peer feedback provides more immediate feedback, helps the students to understand their own work better, and improves student attitudes towards the learning process [21].

*Software Testing* is an evaluation methodology for which developers prepare test cases that exemplify how a program should behave. The practice is widely used in industry, and is useful for attaining high program correctness. Testing is therefore an important skill for students to learn but is often sidelined in computing curricula.

This paper presents the Peer-Testing learning activity and tool we have developed which structures peer feedback with software testing. We deployed Peer-Testing in a second-year computer science data structures and algorithms university course and evaluated the attitude of students towards Peer-Testing to identify its benefits and drawbacks as a learning activity. Although we had initially developed Peer-Testing as a standalone Web platform [9, 17], we have

bridged its functions with a GitLab instance we use for students to manage their coursework source codes. We consider here the following research questions:
**RQ1** Does testing-based peer feedback show the same evidence of deeper learning that peer feedback does in the literature?
**RQ2** Does testing-based peer feedback improve student's perceptions of, and ability in, testing?
**RQ3** Do tests provide an effective structure for peer feedback?

### 1.2 Contributions and Plan

The main contributions of this paper are as follows.

- Definition of the Peer-Testing learning activity and overview of its Web platform for software testing-based peer feedback.
- Presentation of a mixed methods approach to evaluate the impact of Peer-Testing on students' learning and perspective on software testing.
- Report on Peer-Testing evaluation suggesting that it retains the positives found in regular peer feedback, that testing has a valuable impact in framing peer interactions, that it has a positive impact on students' perception of software testing, and it also has potential drawbacks.

The paper presents in Section 2 the Peer-Testing learning activity we have developed. It presents in Section 3 its mixed methods evaluation in the context of a deployment in a course. Related works are discussed in Section 4.

## 2 PEER-TESTING LEARNING ACTIVITY AND WEB PLATFORM

In this section we first present the learning activity we have developed and then its Web platform implementation in Section 2.3. The activity is composed of two main stages which are detailed in Sections 2.1 and 2.2 below and illustrated in Figures 1 and 2.

In preparation for the activity, the teacher creates a coursework exercise. For simplicity, we will be considering a Java based coursework, but the activity and Web platform are suitable for other programming languages. The coursework requires students to implement a solution to a problem and to test solutions to the problem. For this purpose, an *interface* needs to be provided which the solution and tests must adhere to. In addition to this interface, the teacher should provide sample test cases and an oracle solution. The students will be able to use the *sample test cases* to check that their solution matches the interface and elements of the coursework specification. The *oracle solution* will act as a reference implementation that students can use to check their own test cases. The actual source code of the oracle solution is however not visible to the students. A single coursework can contain multiple interfaces, tests cases or oracle solutions (such additional solutions could also be presented as erroneous solutions to help students construct effective test cases).

### 2.1 Self Testing Stage

During this first stage illustrated in Figure 1, students are working individually on their programming solution and their test cases. They get the opportunity to run tests on their development either on their IDE or with the continuous integration feature of the GitLab instance which hosts their code. In addition, the Peer-Testing Web platform allows them to run their test cases on an oracle solution provided to gain an understanding of the implementation requirements. These tests are performed in a black box way as the students do not see the source code of the oracle. This stage is essential for a couple of reasons. It familiarises the students with the Peer-Testing Web platform, it gives them a first feel of running their test cases on another solution than theirs, and it makes their code readily available on Peer-Testing for when Peer-Testing is moved to the next stage.

In our setting, initial source code skeleton files for the coursework are made available in a project on our GitLab instance. The students have to fork the project to their own GitLab space. The students are prevented from sharing their forked project in this context. We made the choice of requiring the student to actively fetch their code from GitLab to Peer-Testing,to make them conscious of the version being tested.

This stage ends with the submission deadline when the platform is moved to peer testing mode.

### 2.2 Peer Testing Stage

Prior to this stage, the teacher will have created *peer groups* of students on the platform. After the submission deadline, the teacher moves the platform into peer testing mode allowing students to immediately run tests on solutions of their peers.

Alongside the coursework, marks are awarded as an incentive to participate in the formative feedback activity but are not dependent on the quality or quantity of interactions a student has.

In addition to running test cases on their own solution or on the oracle solution, they can now do so on their peers' solutions as illustrated in Figure 2. Once a test is run, the results are displayed, and the test and solution are made visible to both students. For each test run, the two students involved can add comments to a feedback panel allowing for a discussion to take place. The starting point of each discussion is therefore the result of a test run which has the advantage, as the result of a computation, of being non-judgemental information.

This part of Peer-Testing takes place on the Web platform. At any point, the students can fetch newer versions of their source code and tests from GitLab. If a student was inclined to do so, they could make changes to their solution after a discussion with a peer, but this is not a requirement.

*Anonymity.* Students can see other group members in an anonymous fashion, with peers referred to as "Peer #1", "Peer #2", etc. The students are instructed in the earlier stages not to include identifying information in the solutions and tests they create. Anonymity has been shown to improve student performance in peer assessment [14]. The teacher can see all of the interactions between students on the platform with the actual names, so anonymity is not preserved in the eyes of the teacher, only between peers.

*Peer Groups.* Given many peer solutions to review, it was found [16] that computer science students preferred doing them all at once, in parallel. However, this study also found that performing each one individually resulted in more useful critical comments

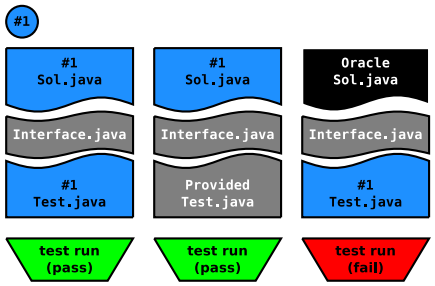


**Figure 1: Self Testing Stage**

*A student #1 develops a solution and test cases. The platform allows running test cases on a solution (as students would do on their IDE) and in addition on an oracle solution provided (without access to its sources).*

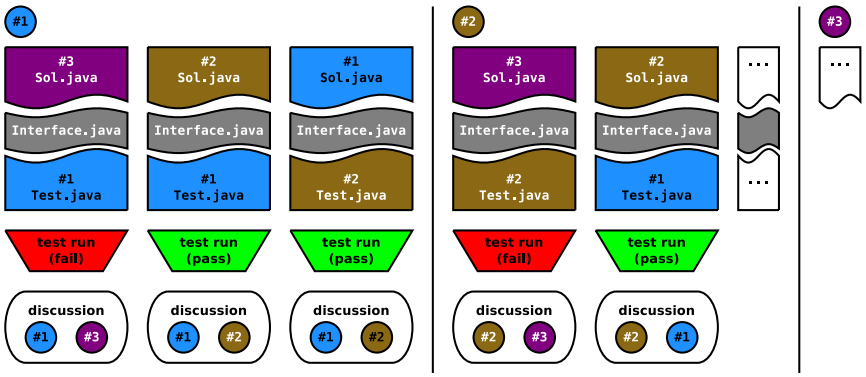


**Figure 2: Peer Testing Stage**

*After the submission deadline, the students can create test runs with their peers. Each student is enrolled in a peer group and can initiate test runs opening one-to-one discussions. A student #1 runs her test cases on peer #3's solution. The test run's result is available to both students as well as a discussion panel for the test run. For each peer pair, the students can run test cases on the peer's solution or run the peer's test cases on their own solution. In each case, the result is available to both students with a discussion panel. They have access to their peer's source code after a run was initiated.*

being produced. Peer-Testing handles reviews using unit testing as a base, favouring sequential review of peer solutions.

There is more benefit to students when they complete more than one review. A study [18] found that when compared to a standard evaluation, aggregated peer reviews are accurate in finding feedback for a given solution. Peer-Testing uses groups to ensure each student receives multiple reviews.

*Final Feedback and Reflective Summary.* The last part of the activity does not involve the platform and is completed through the usual assessment submission system. At this point, the students are to complete a reflective summary about their interactions in Peer-Testing. This summary should address feedback from peers, observations made, or improvements considered.

## 2.3 Web Platform with GitLab Bridging

The Peer-Testing Web platform comprises a custom website developed with Django[1] to manage test runs and peer interactions. It currently supports Java and Python submissions, but is generic enough that it could be extended to other languages. Users of the platform are authenticated as either a student or a teacher. The platform makes use of an external GitLab[2] instance for version control, user authentication and user notification (see Figure 3). However, the Django website has its own user authentication system and could accept student submissions with Web forms. Test runs are managed by the Django website although testing job management could be transferred to GitLab in future versions of the platform. We mitigate security concerns associated with running uploaded code by limiting the memory space and duration of execution and by controlling the execution with Java's Security Manager. The Web platform is publicly available [3] under a free software licence.

[1]https://www.djangoproject.com/
[2]https://about.gitlab.com/
[3]https://github.com/peergramming/peer-testing

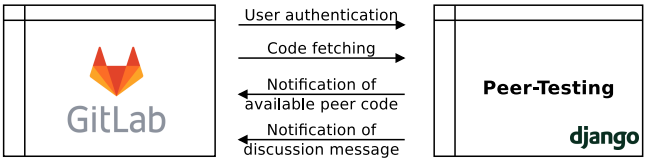


**Figure 3: General Architecture of Peer-Testing Web Platform**

*The Peer-Testing Web platform is Django-based and couples with a GitLab instance to facilitate user authentication, source code management and user notification. The platform could also be used as a standalone website with Django's user authentication system and with Web forms to upload code.*

# 3 EVALUATION

In this section we discuss the evaluation of Peer-Testing using a mix of methods (Section 3.1) based on a deployment of the activity as part of a computer science course at the University (Section 3.2). The evaluation analysis (Section 3.3) is structured along the research questions we introduced in Section 1.1.

## 3.1 Mixed Methods Approach

To evaluate Peer-Testing, we decided to use an approach based on a mix of quantitative and qualitative methods. These methods employ information participants consent to provide for the purpose of this research in line with our University's ethical requirements: answers to two questionnaires, the reflective summary submitted for the purpose of the course, the source code versions on the University GitLab, and the peer discussions.

*3.1.1 Participants Questionnaires.* All students were invited to take part in this research. The voluntary participants were asked to fill out a first questionnaire prior to the Peer-Testing activity, and a second questionnaire at the end of the activity. The two questionnaires share some questions for pre/post evaluation (see Figure 5).

Additional questions about the Peer-Testing activity specifically were included in the second questionnaire (see Figure 4).

*3.1.2 Participants' Reflective Summaries.* We coded the reflective summaries that the study participants submitted. Students received marks for taking part in peer testing and for submitting a reflective summary but, as the marks were purely an incentive to take part, the marks were not given according to the level of engagement nor content of the summary. The coding took place after the marking was concluded.

For each reflective summary, we looked for the presence of negative, mixed, and positive feelings. For positive feelings, we separated the impact on learning into a generic surface learning code, and deeper learning code when the summary included evidence that the activity triggered critical thinking. This coding was done across 4 categories: giving and receiving feedback, for both programming and testing. Coding was performed by two coders who first independently coded a common sample of summaries and then agreed on the coding criteria.

*3.1.3 Participants' Source Code Changes and Discussions.* We analysed the git commits that were made after the coursework deadline, once the peer testing stage had started, and looked for relations to discussions that took place on the platform. We then analysed the kind of changes that were being made. In case of a commit after the deadline, students would not receive extra marks for improvements. The fact that some students made post-deadline changes may indicate that Peer-Testing interactions were a motivating factor for updating their code.

## 3.2 Context of Evaluation

We focused our evaluation on one substantial piece of coursework the students had four weeks to develop, followed by two weeks of Peer-Testing peer testing stage.

## 3.3 Analysis of Results

A third of the students agreed to participate in this study. These 81 participants are a relatively good representation of the distribution of grades in the cohort except for an inflated representation of A grades as shown in Table 1. The pre-questionnaire was answered by 43 and the post questionnaire by 57 of which 19 participants answered both. Among these participants, 66 submitted a reflective summary which we coded. Although the number of participants is relatively low, employing a mixed methods approach should give interesting insights into the learning activity.

***RQ1*** *Does testing-based peer feedback show the same evidence of deeper learning that peer feedback does in the literature?* The coding of reflective summaries indicates that a third contain evidence of deeper learning on programming compared to about a fifth on testing. We split this coding per grade obtained by the participants in this course to be able to compare our observation. Table 1 gives the split per grade and indicates with + signs when coding resulted in a higher percentage of summaries than overall. Strong students tend to show more evidence of deeper learning and tend to mention testing more often in their summaries. Weaker students are reporting fewer negative feelings about the activity.

| | A | B | C | D | E | F | Total |
|---|---|---|---|---|---|---|---|
| Cohort | 67 | 34 | 28 | 36 | 18 | 39 | 222 |
| Participants | 34 | 13 | 9 | 8 | 8 | 9 | 81 |
| % of cohort | 50% | 38% | 32% | 22% | 44% | 23% | 33% |
| Summaries | 31 | 12 | 8 | 6 | 5 | 4 | 66 |
| % of cohort | 46% | 35% | 28% | 16% | 27% | 10% | 29% |
| Codes related to programming/testing | | | | | | | 82/42 |
| - deep | +/+ | +/+ | –/– | ·/+ | –/– | –/– | 33%/18% |
| - surface | ·/+ | –/– | –/– | +/· | +/– | –/– | 58%/32% |
| - mixed | ·/+ | –/– | +/– | ·/– | –/– | +/– | 15%/6% |
| - negative | ·/· | +/– | ·/+ | –/– | –/– | –/+ | 9%/6% |

**Table 1: Participants and Reflective Summaries Coding per Grade (fall 2019)**

*The table gives: the distribution of grades for the course across the cohort and the subset of participants; the overall number of codes assigned and the percentage of participants' reflective summaries with code; the variation of participation and of codes distribution per grade using colours and – or + signs.*

In a reflective summary, one student makes a point of showing how they initially thought testing was purely a tool for themselves, but now sees how it can be used in a more general software development capacity. This is a positive outcome to see, which might not have happened with a regular peer review exercise with no testing:

> *"It had not occurred to me to use one's test cases against someone else's code, [...] this has given me a better insight into how in-depth code review can be used in industry".*

Another important aspect of peer feedback is timely feedback. Peer-Testing is objectively faster than teacher-produced feedback, as peers can interact soon after the coursework deadline. However, there was mixed feedback as some peer groups with low participation led to students feeling as if they were delayed or that the exercise was less useful:

> *"It is important to have a very active participation from all participants to have a truly worthwhile experience".*

Others appreciated the back-and-forth discussions:

> *"Peer#6 was a genius. [...] I learnt a lot from him. Also asked him few questions which helped me".*

The answers to the post questionnaire, shown in Figure 4, confirm the common positive view students have on peer feedback activities (with a slight preference to giving than receiving feedback) and that such activity help in understanding the coursework differently.

Another similarity with other form of peer feedback is on erroneous or misleading feedback students could exchange. We observed three cases of incorrect feedback which were reported and questioned in the reflective summary by the student receiving the feedback.

***RQ2*** *Does testing-based peer feedback improve student's perceptions of, and ability in, testing?* Looking at the evolution of pre and post responses on the likelihood of using testing, shown in Figure 5, there is an increase in students responding they would use testing

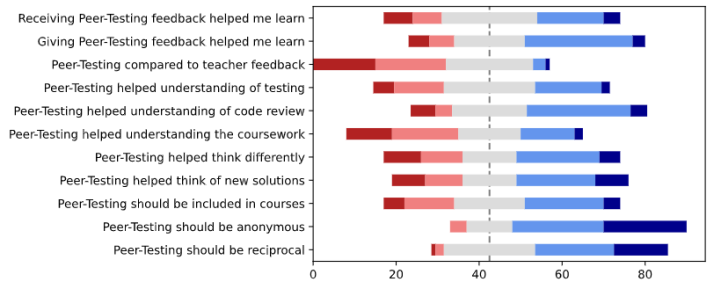


**Figure 4: Post Activity Answers to Questionnaire (# Responses)**
*Likert 5-scale answers (strongly disagree in dark red to strongly agree in dark blue).*

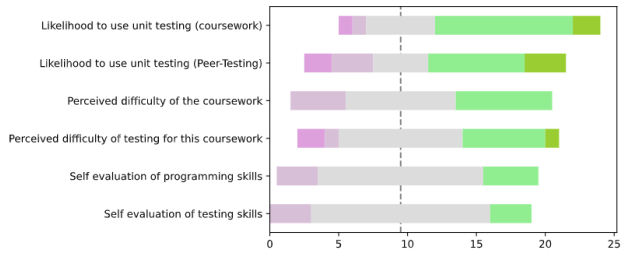


**Figure 5: Evolution Pre/Post Activity Answers (# Changed)**
*Difference in Likert 5-scale answers (-2 in dark purple to +2 in dark green).*

in the future. However, there is a stronger feeling that this is due to the actual coursework rather than Peer-Testing.

From the reflective summaries we found evidence of changed attitudes, given an increase in the likelihood to use testing in the future (Figure 5). One student noted that the activity gives an insight into peers' mindsets:

> *"Peer-Testing taught me much more about my peers and the way they think than anything code related".*

This could be a desirable outcome as it helps broaden student perspectives and viewpoints. This ability to see things from different perspectives is helpful when developing and evaluating systems.

Students felt that Peer-Testing was not an adequate replacement for teacher feedback (Figure 4), and there were mixed responses about whether Peer-Testing should be included in courses. Peer-Testing had a mixed impact on self-evaluation of coding/testing (Figure 5), however, 63% of the 19 participants who answered both questionnaires received A grades, so these results may be biased.

***RQ3*** *Do tests provide an effective structure for peer feedback?* To investigate this question, we looked at the interactions that occurred between peers and in particular if and how peer discussion triggered change to students' source code. We observed three types of commits being made affecting solution, test cases, and anonymity (three students removed names from comments suggesting that students within a peer feedback framework do seek to remain anonymous).

Six students amended their JUnit test cases after the coursework submission deadline. In one instance a student changed their test to remove test cases specific to their own implementation. In another case, a student tester attempted to "fix" a test case which was not passing for one solution, indicating active attempts to understand the cause.

> *[Peer #1] For some reason test 7 is failing where others have managed to pass it so it most likely isn't an issue with the tests or others may be doing it wrong and my test is incorrectly written.*

Students were provided with a simple test case, and more complex test cases for self testing before the coursework deadline. Several students uploaded just the simple test to Peer-Testing before later committing the more complex one. Indicating that Peer-Testing prompted them to want to do more testing. Some 7 students updated their source code after the coursework submission deadline. There is evidence that discussion of test results encouraged students to attempt to fix their program. Two peers attempted an experiment on the efficiency of a polynomial accumulation used within a hashing function:

> *[Peer #1] I used 31 in my code. I see that you have chosen to use 32. I was intrigued as to why java implementation uses 31 instead of 33 or 32. [...] It would be awesome if you too could try out with 31 instead of 32*
> [The two peers then exchange results of varying times taken to run the program when different modulus values are used]
> *[Peer #1] To conclude: I feel 31 and 32 are significantly better options than 33. [...] Anyway, thank you soo much for the time you took to do the tests!*
> *[Peer #2] No worries [...] it was an interesting experiment*

Through curiosity arising from the discussion, they changed source code and experimented with tests to try to make their implementation more efficient. This is a very positive outcome to see.

# 4 RELATED WORKS

## 4.1 Peer Review

Student peer review systems could be used in a programming course context. A report [25] of such experiment using the PRAZE system noted that the process should "develop important skills in the reading and understanding of other people's code". While PRAZE was not designed with programming code in mind, we built Peer-Testing to facilitate the sharing of codes, tests, and test runs sharing. Peer-Testing enables back-and-forth discussion which was a feature of over half of the peer review platforms a survey analysed [15].

*Peer Code Review (PCR).* PCR refers to peers accessing and reviewing each other's code with the aim of providing constructive feedback. This commonly happens in industry and is also used in classroom environments. Peer-Testing incorporates elements of PCR, by getting users to share feedback about code. However, it differs from PCR by basing the review on the results of test runs. Multiple studies [12, 23] which found PCR to be helpful early on in computer science undergraduate courses suggest some best practices when engaging in PCR such as establishing review proforma, mixing levels in peer groups, or applying anonymity. These studies have found that PCR improved attitudes in students, honed critical review skills and improvements to teamwork and communication.

Our investigation has shown similar outcomes with Peer-Testing. Despite re-submission after review being common in many studies [11, 26, 28], we require submission of a reflective report rather than re-submission of code to reduce the students' workload and focus their effort on critically analysing their work. The students have however the possibility to submit newer versions of their solution and tests during the Peer-Testing stage to extend the possibility of a back-and-forth peer discussions. Another difference with most PCR approaches is that Peer-Testing produces a set of one-to-one interactions rather than a team or group review.

*Peer Review and Anonymity.* An experiment into peer review to teach software testing ensured a double-blind nature of code submission by obfuscating source code into byte code [23]. Though anonymity is heightened (e.g. identifying comments are removed), it does mean students would have a harder time reading the code and would not be able to review coding style. Another experience report [26] reveals similar issues with this kind of double-blinding method. To that end we decided not to obfuscate code in Peer-Testing, and simply asked students to not leave any identifying comments, and kept anonymity by not showing names when sharing reviews and code.

## 4.2 Software Testing in Peer Interactions

Many classrooms have made use of exercises which have similarities to Peer-Testing. Some of these will be detailed in this section, as well as noting some key differences introduced by Peer-Testing.

While running student-written software tests across other students' solutions is not a new concept [8], unit tests (such as JUnit in Java) have problems in that they may not be compatible across different students' solutions. Research has suggested the use of dedicated tools [5] to resolve compatibility issues, whereas we decided to require that students follow a specified Java interface and only test against that interface. Peer-Testing could help students understanding that a unit test can be specific to an implementation or general to a specification (see Section 3.3).

Reporting on a software testing system [24], the authors suggest using a training tool where students write tests against a set of buggy programs, and describe the benefits to students' abilities in writing black-box tests. Peer-Testing combines testing potentially buggy programs created by students with having the knowledge implementing such program.

Testing of peer code have been conducted as part of a software engineering undergraduate course [3] which included code review and "Group peer inspections". These involved partnered students providing feedback on code projects through code review and informal testing. Similar to Peer-Testing, this exercise uses code review as a medium for the feedback given, and it is performed on actual coursework code produced by students. Other than the individual approach we take, a major difference is that we explicitly require the use of unit testing throughout the completion of the coursework, and use that as a base for the code review stage. This approach also intersperses peer testing throughout a coursework task rather than just at the end.

The *Praktomat* system [28] is a platform to enable PCR, but similar to Peer-Testing, submitted code must first compile and pass tests before being distributed for peer review. Producing appropriate tasks and tests presents a lot of work for the teachers beforehand, though the automatic testing does reduce the workload during submission. So that most of the testing effort is done by the students, the test cases we provided for the Peer-Testing activity are basic (they mainly check that a solution conforms to the interface).

*Gamification.* Gamification is an increasingly used technique in programming education. Games involving developing a solution according to a specification have been developed *Pex4fun* and the later *Code Hunt* [1, 27] and a game that teaches mutation testing, Code Defenders was created [4, 20]. The Peer-Testing process could be adapted to include gamified elements to increase student engagement.

## 4.3 Automatic Assessment and Feedback

There exist many uses of automatic assessment in programming education. The main aim of Peer-Testing is to enable the students to run tests, but automated assessment techniques could be incorporated into Peer-Testing to guide further the students' peer reviews and to assist teachers in grading assignments.

*Suggestion Mining.* Suggestion mining is the practice of analysing program code and the progress made by students in their assignments, and then using this to offer suggestions to students on how to proceed next. Suggestion mining tools such as *MistakeBrowser* and *FixPropagator* have been used in the context of a course [10]. The former system could identify problems for students, and the latter allowed teachers to show a correction for common clusters of mistakes which could automatically be propagated to the contexts of the solutions of students. This kind of workflow could be integrated into the platform so that teachers or potentially students could quickly address issues in code when giving feedback.

# 5 CONCLUSION AND FUTURE WORK

In this paper we defined Peer-Testing, an approach to base peer feedback on software testing. We described the stages of the learning activity and the features of the Web platform we develop to support it. We evaluated the learning activity in a higher education context using a mixed of methods. The results of the evaluation confirm that Peer-Testing retains the benefits that peer feedback activities have. In addition, the impact on students' understanding and on their perception of testing is evidenced. Peer-Testing however seems to trigger a deeper understanding of testing only within stronger students. As future work, we aim to incorporate more guidance within the system in particular to assist weaker students. Peer-Testing could be combined with information and support for test coverage for instance. The quality of the tests students write would become a more prominent focus of the activity and its impact would need to be evaluated. The evaluation we reported on was based on a relatively small number of participants. We aim to develop and evaluate Peer-Testing further by deploying the activity to other kind of courses involving programming, and by broadening the scope of the evaluation with more participants and a longitudinal view. On the technical side, we would like to increase the GitLab integration so that test runs jobs are managed within GitLab.

# REFERENCES


[1] Judith Bishop, R. Nigel Horspool, Tao Xie, Nikolai Tillmann, and Jonathan De Halleux. 2015. Code Hunt: Experience with Coding Contests at Scale. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, Vol. 2. 398–407. https://doi.org/10.1109/ICSE.2015.172

[2] David Carrington. 1996. Teaching Software Testing. In *Proceedings of the 2nd Australasian Conference on Computer Science Education (ACSE '97)*. ACM, New York, NY, USA, 59–64. https://doi.org/10.1145/299359.299369

[3] Nicole Clark. 2004. Peer Testing in Software Engineering Projects. In *Proceedings of the Sixth Australasian Conference on Computing Education - Volume 30 (ACE '04)*. Australian Computer Society, Inc., Darlinghurst, Australia, Australia, 41–48. http://dl.acm.org/citation.cfm?id=979968.979974

[4] Benjamin S. Clegg, José Miguel Rojas, and Gordon Fraser. 2017. Teaching Software Testing Concepts Using a Mutation Testing Game. In *Proceedings of the 39th International Conference on Software Engineering: Software Engineering and Education Track (ICSE-SEET '17)*. IEEE Press, Piscataway, NJ, USA, 33–36. https://doi.org/10.1109/ICSE-SEET.2017.1

[5] Stephen H. Edwards, Zalia Shams, Michael Cogswell, and Robert C. Senkbeil. 2012. Running students' software tests against each others' code: new life for an old "gimmick". In *Proceedings of the 43rd ACM technical symposium on Computer Science Education - SIGCSE '12*. ACM Press, Raleigh, North Carolina, USA, 221. https://doi.org/10.1145/2157136.2157202

[6] Nancy Falchikov. 2005. *Improving Assessment Through Student Involvement: Practical Solutions for Aiding Learning in Higher and Further Education*. Routledge.

[7] Samuel B. Fee and Amanda M. Holland-Minkley. 2010. Teaching computer science through problems, not solutions. *Computer Science Education* 20, 2 (June 2010), 129–144. https://doi.org/10.1080/08993408.2010.486271

[8] Michael H. Goldwasser. 2002. A Gimmick to Integrate Software Testing Throughout the Curriculum. In *Proceedings of the 33rd SIGCSE Technical Symposium on Computer Science Education (SIGCSE '02)*. ACM, New York, NY, USA, 271–275. https://doi.org/10.1145/563340.563446

[9] Gudmund Grov, Mohammad Hamdan, Smitha Kumar, Manuel Maarek, Léon McGregor, Talal Shaikh, J. B. Wells, and Hind Zantout. 2017. Transition from Passive Learner to Critical Evaluator through Peer-Testing of Programming Artefacts. *New Directions in the Teaching of Physical Sciences* 0, 12 (Nov. 2017). https://journals.le.ac.uk/ojs1/index.php/new-directions/article/view/2398

[10] Andrew Head, Elena Glassman, Gustavo Soares, Ryo Suzuki, Lucas Figueredo, Loris D'Antoni, and Björn Hartmann. 2017. Writing Reusable Code Feedback at Scale with Mixed-Initiative Program Synthesis. In *Proceedings of the Fourth (2017) ACM Conference on Learning @ Scale (L@S '17)*. ACM, New York, NY, USA, 89–98. https://doi.org/10.1145/3051457.3051467

[11] Christopher Hundhausen, Anukrati Agrawal, Dana Fairbrother, and Michael Trevisan. 2009. Integrating Pedagogical Code Reviews into a CS 1 Course: An Empirical Study. In *Proceedings of the 40th ACM Technical Symposium on Computer Science Education (SIGCSE '09)*. ACM, New York, NY, USA, 291–295. https://doi.org/10.1145/1508865.1508972

[12] Christopher D. Hundhausen, Anukrati Agrawal, and Pawan Agarwal. 2013. Talking About Code: Integrating Pedagogical Code Reviews into Early Computing Courses. *Trans. Comput. Educ.* 13, 3 (Aug. 2013), 14:1–14:28. https://doi.org/10.1145/2499947.2499951

[13] Essi Lahtinen, Kirsti Ala-Mutka, and Hannu-Matti Järvinen. 2005. A Study of the Difficulties of Novice Programmers. In *Proceedings of the 10th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education (ITiCSE '05)*. ACM, New York, NY, USA, 14–18. https://doi.org/10.1145/1067445.1067453

[14] Lan Li. 2016. The role of anonymity in peer assessment. *Assessment & Evaluation in Higher Education* (April 2016), 1–12. https://doi.org/10.1080/02602938.2016.1174766

[15] Andrew Luxton-Reilly. 2009. A systematic review of tools that support peer assessment. *Computer Science Education* 19, 4 (Dec. 2009), 209–232. https://doi.org/10.1080/08993400903384844

[16] Andrew Luxton-Reilly, Arthur Lewis, and Beryl Plimmer. 2018. Comparing Sequential and Parallel Code Review Techniques for Formative Feedback. In *Proceedings of the 20th Australasian Computing Education Conference (ACE '18)*. ACM, New York, NY, USA, 45–52. https://doi.org/10.1145/3160489.3160498

[17] Manuel Maarek and Léon McGregor. 2017. Development of a Web Platform for Code Peer-Testing. In *The 8th Workshop on Evaluation and Usability of Programming Languages and Tools (PLATEAU) at SPLASH 2017*.

[18] Ken Reily, Pam Ludford Finnerty, and Loren Terveen. 2009. Two Peers Are Better Than One: Aggregating Peer Reviews for Computing Assignments is Surprisingly Accurate. In *Proceedings of the ACM 2009 International Conference on Supporting Group Work (GROUP '09)*. ACM, New York, NY, USA, 115–124. https://doi.org/10.1145/1531674.1531692

[19] Anthony Robins, Janet Rountree, and Nathan Rountree. 2003. Learning and Teaching Programming: A Review and Discussion. *Computer Science Education* 13, 2 (June 2003), 137–172. https://doi.org/10.1076/csed.13.2.137.14200

[20] José Miguel Rojas, Thomas D. White, Benjamin S. Clegg, and Gordon Fraser. 2017. Code Defenders: Crowdsourcing Effective Tests and Subtle Mutants with a Mutation Testing Game. In *Proceedings of the 39th International Conference on Software Engineering (ICSE '17)*. IEEE Press, Piscataway, NJ, USA, 677–688. https://doi.org/10.1109/ICSE.2017.68

[21] Philip M. Sadler and Eddie Good. 2006. The Impact of Self- and Peer-Grading on Student Learning. *Educational Assessment* 11, 1 (Feb. 2006), 1–31. https://doi.org/10.1207/s15326977ea1101_1

[22] Terry Shepard, Margaret Lamb, and Diane Kelly. 2001. More Testing Should Be Taught. *Commun. ACM* 44, 6 (June 2001), 103–108. https://doi.org/10.1145/376134.376180

[23] Joanna Smith, Joe Tessler, Elliot Kramer, and Calvin Lin. 2012. Using Peer Review to Teach Software Testing. In *Proceedings of the Ninth Annual International Conference on International Computing Education Research (ICER '12)*. ACM, New York, NY, USA, 93–98. https://doi.org/10.1145/2361276.2361295

[24] Rebecca Smith, Terry Tang, Joe Warren, and Scott Rixner. 2017. An Automated System for Interactively Learning Software Testing. In *Proceedings of the 2017 ACM Conference on Innovation and Technology in Computer Science Education - ITiCSE '17*. ACM Press, Bologna, Italy, 98–103. https://doi.org/10.1145/3059009.3059022

[25] Harald Sondergaard. 2009. Learning from and with Peers: The Different Roles of Student Peer Reviewing. In *Proceedings of the 14th Annual ACM SIGCSE Conference on Innovation and Technology in Computer Science Education (ITiCSE '09)*. ACM, New York, NY, USA, 31–35. https://doi.org/10.1145/1562877.1562893

[26] Yanqing Wang, Hang Li, Yuqiang Feng, Yu Jiang, and Ying Liu. 2012. Assessment of programming language learning based on peer code review model: Implementation and experience report. *Computers & Education* 59, 2 (Sept. 2012), 412–422. https://doi.org/10.1016/j.compedu.2012.01.007

[27] Tao Xie, Judith Bishop, Nikolai Tillmann, and Jonathan de Halleux. 2015. Gamifying Software Security Education and Training via Secure Coding Duels in Code Hunt. In *Proceedings of the 2015 Symposium and Bootcamp on the Science of Security (HotSoS '15)*. ACM, New York, NY, USA, 26:1–26:2. https://doi.org/10.1145/2746194.2746220

[28] Andreas Zeller. 2000. Making Students Read and Review Code. In *Proceedings of the 5th Annual SIGCSE/SIGCUE ITiCSEconference on Innovation and Technology in Computer Science Education (ITiCSE '00)*. ACM, New York, NY, USA, 89–92. https://doi.org/10.1145/343048.343090