Development of a Web Platform for Code Peer-Testing

Development of a Web Platform for Code Peer-Testing

Manuel Maarek
School of MACS
Heriot-Watt University
Edinburgh, Scotland, UK
M.Maarek@hw.ac.uk

Léon McGregor
School of MACS
Heriot-Watt University
Edinburgh, Scotland, UK

Abstract

As part of formative and summative assessments in programming courses, students work on developing programming artifacts following a given specification. These artifacts are evaluated by the teachers. At the end of this evaluation, the students receive feedback and marks. Providing feedback on programming artifacts is time demanding and could make feedback to arrive too late for it to be effective for the students' learning. We propose to combine software testing with peer feedback which has been praised for offering a timely and effective learning activity with program testing.

In this paper we report on the development of a Web platform for peer feedback on programming artifacts through program testing. We discuss the development process of our peer-testing platform informed by teachers and students.

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

Keywords peer testing, peer feedback, software testing

1 Introduction

Students in Computer Science (CS) courses take part in individual coursework exercises in the form of programming tasks. Such programming coursework often play an important role in their assessment and are an opportunity to train and develop the students programming skills, as such the feedback they receive on the artifacts they develop is important for their learning. While students receive face-to-face feedback on their programs during computer lab sessions, the bulk of feedback accompanies the grading of the artifacts they have submitted for assessment. As producing programming feedback is a time consuming task, a number of experiments have been conducted to use testing of programming artifacts to automatically produce feedback to students where failed and passed tests construct the program feedback. In CS and Higher Education (HE) in general, peer feedback which involves students providing feedback on their peers' work is often praised for offering a timely and effective learning activity.

We propose to combine peer feedback and software testing to offer timely feedback on programming artifacts. In this approach, after submitting their own programming artifacts, students would engage in a peer feedback activity on their peers' submissions. The main medium for feedback is software tests that students would run on their peers' submissions. To facilitate this peer feedback activity, we have developed a peer-testing Web platform which manages the submission of solutions and tests, their execution, and the sharing of results and textual feedback. Ideally, the peer feedback that takes place within the Web platform will provide more immediate and useful feedback on coursework than having to wait for a teacher, and the involvement in the peer-testing activity invites students to engage in critical thinking about the programming process.

This paper reports on the development of our peer-testing platform which was informed by teachers and students. The paper:

- Presents peer-testing as peer feedback on programming artifacts through program testing.
- Discusses aspects and features teachers expected from a peer feedback system for code peer-testing.
- Report students opinions on the Web platform after experiencing peer-testing.
- Describes the peer-testing platform we have implemented and the stages making a peer-testing activity.

Plan In Section 2 we give the background of this work on peer feedback, peer assessment and software testing. We describe in Section 3 how we derived the key requirements for the peer-testing platform from discussions with teachers and students, and we give an overview of the platform we have implemented in Section 4. We then discuss related works in Section 5 before drawing perspectives for this work in Section 6.

The work presented here took place at Heriot-Watt University during Léon McGregor's BSc Honours project [McGregor 2017] and as part of the University's Learning and Teaching Enhancement project lead by Manuel Maarek. Aspects of the project, in particular its impact on students'

PLATEAU'17 Workshop on Evaluation and Usability of Programming Languages and Tools, October 23, 2017, Vancouver, CA

transition from passive learners to critical evaluators was presented at Horizons in STEM Higher Education 2017 Conference [Grov et al. 2017].

2 Background

2.1 Peer Feedback and Peer Assessment

Peer assessment is a process in which students assess each other. It is in contrast to the more traditional stance where a teacher performs the assessment. As defined by Topping [2009],

> "Peer assessment is an arrangement for learners to consider and specify the level, value, or quality of a product or performance of other equal-status learners."

That is to say, students with a similar level of education assessing the work of each other to give critical feedback and discussion. This could be done in many ways, such as between pairs or in groups, and can be performed on any number of different activities from programming exercises to oral reports.

Peer assessment is a process with many benefits to participants in education. Sadler and Good [2006] have suggested the following concepts that peer assessment can help with

- Peer assessment is more immediate, so students can get more feedback, and sooner;
- Students performing marking can reduce the workload for teachers;
- The process of checking and thinking about another students answer can improve a students own understanding;
- Peer assessment can help students better understand testing and can become aware of their own strengths and weaknesses;
- Following peer assessment students can gain an improved attitude towards the process of learning as a whole.

Peer assessment can offer much help towards education of students, but it would be worthwhile to know just which aspects are the most useful. A study conducted by Li et al. [2010] investigated the peer assessment process with the aim of discovering which part of it is most useful to the students involved: being an assessor or being assessed. To study this, undergraduate student teachers were given the task of creating a WebQuest project (activities for student to learn from Internet resources) This was then marked by independent assessors, and the student teachers were given a chance to provide feedback on other student teachers' WebQuests. Following this, the feedback was returned and student teachers were given another chance to improve their project, and it was marked again. The quality of the peer feedback itself was also checked by the independent markers. The study found that

> "there was a significant relationship between the quality of the peer feed-back the students provided for others and the quality of the students' own final projects"

The findings of the investigation would suggest that the actual exercise of providing feedback to others (acting as an assessor) is a worthwhile process for learning from. This study also concluded that there was no reasonable link between the feedback itself as a learning tool, suggesting that the act of giving feedback itself is more valuable and that low quality feedback does not harm the learning experience.

With the knowledge that peer assessment can be useful, it is important to know how a peer assessment should be conducted. A study performed in a classroom environment by Smith et al. [2012] focused more on the use of peer assessment as a tool for teaching testing of code, in addition to the existing course. Over the course of this study, which took place using coursework from a 12-week university course, the following was completed for each coursework: submitting solutions, then submitting peer reviews (which includes a description of the testing that they performed on another solution, and the results of this), and then a review of the peer assessment (including what was learnt, an evaluation of feedback on their own solution, and optionally a corrected solution). One particularly noteworthy aspect of this use of peer assessment was the double-blind nature, ensuring anonymity. Students would not be aware of who they were marking, or were marked by. To enforce this completely, submitted code was obfuscated (Java sources into Byte code). One advantage of this is that it strips out identifying variable names and comments, which could identify other students. However, a downside of using byte code is that it can make it difficult to do in-depth analysis of the source structure which may make it harder to write complete test cases. The study identified two key features that assignments for peer assessment need to have: a well-defined interfaces, and freedom for implementation. In addition to this, Smith et al. [2012] has found that it was possible to integrate the peer assessment process without having to significantly alter the existing course material, and the students taking part enjoyed the experience. This shows promise, as it could indicate many CS courses (that offer coursework meeting the requirements), could be modified to include their own peer assessment exercises.

Peer assessment can prove to be a very valuable experience for students. Falchikov [2013] has collected various case studies of past peer assessments, and the following aspects can be found:

- If the marking criteria are properly explained, there is often no significant difference between the marks awarded by students and those that would be awarded by teachers. This would tend to indicate that students do assess each other fairly.

- One of the most important aspects of peer assessment is the ability of the student to learn how to assess other students and from this learn how to critically assess and improve their own work.
- It is important to make sure students feel confident, otherwise they may not assess their peers as honestly as they might otherwise have done. Some students will feel conflicted about marking their peers, particularly if they might have to give low marks.
- During peer assessment more benefits may come from students assessing multiple solutions, rather than each focusing on one.

2.2 Software Testing Methods

When considering the testing of software, there are several ways that this can be done. To produce a Web platform that enables peer assessment, an appropriate testing methodology will need to be selected. Based on suggested testing styles from Laboon [2016], some testing methodologies were considered for inclusion.

Linting Performing basic checks for 'smells' of bad code design, such as identifying unused variables or methods. Not particularly useful from a perspective of checking code is correct, but is still useful in terms of producing good quality code.

Unit testing The use of xUnit style tests. Often used in Test-Driven Development (TDD), these could be useful in detecting flaws if written post-development by a peer assessor. This would need the assessor to be familiar with how to write xUnit style tests, and how they should be structured, or it would not be viable. (The prototypes and current implementation presented in this paper use the unit testing form of testing)

Expected output testing Running a program with some input and comparing the output to what is expected. The issue here is that the peer assessor first needs to know what the correct output is. Unlike unit testing, this would simply involve a series of inputs, and a series of expected outputs, and the task of checking the correctness from these tests is left to the website.

Scenario Testing Simulate an actual usage scenario. Assessors would use solutions as if they were third party libraries (a *black box*), and develop their own programs that would use these as if in an actual use case. These programs would perform checks to make sure the solutions being tested were running as expected. This is more involved than unit or expected output testing, as it might be better placed to discover side effects of continued use of the solution.

Property based testing Using a proof checker or random testing such as QuickCheck to ensure that a program is acting correctly. This would place additional overhead onto the students as assessors, as they would need to learn and understand the annotations used by a proof checker, but could provide a lot of coverage of possible inputs.

3 Informed Peer-Testing Requirements

The peer-testing platform is a tool for enabling peer-testing of program artifacts as a learning activity in the classroom. As such its main users are the teachers and the students. We have planned the development of the platform into steps involving teachers and students to gathered information about their needs and perspectives on a platform for peer-testing. At each steps we developed a prototype that was used as a base for discussion. In the first step, we implemented an initial prototype following a concise set of requirements (see Section 3.1). In a second step teachers were invited to discuss these requirements and propose new ones (see Section 3.2). A second prototype was then developed implementing some of these requirements, and was evaluated by a pool of students who suggested improvements (see Section 3.3). The current version of the peer-testing platform presented in Section 4 implements some of these suggestions.

3.1 Initial Requirements

In our initial design, the peer-testing process was asynchronous where each student would (1) submit a solution as a source code file of their program. The system would then (2) pair the students into tester/developer before it enters in peer-testing mode, where each student would (3) as tester submit tests, which are run against their peer's solution. Each student would then (4) submit a test with textual feedback which the (5) student as developer would see displayed. As part of the coursework tasks, each student would submit a report discussing their involvement in peer-testing and how they would update their program to take into account the feedback they received or gave. This initial simple workflow was implemented and used as a base for discussion with teachers.

3.2 Focus Group Discussions with Teachers

To get information on the teachers' perspective on peer-testing, we invited seven CS lecturers from our department to discuss the initial set of requirements and to propose features they consider to be essential for a peer-testing platform in the classroom. We summarise here the main topics of discussions.

Self Testing To make sure students are ready to interact with peer-testing it is essential that they test their own implementation. So the platform should allow testing own implementations form the start (we have possible in Stage 1, see Section 4). The ability to craft tests based on someone else's implementation could also be allowed prior to peer-testing but would require to hide the source code of the peer's implementation (black-box testing) to avoid plagiarism issues (we

have made this possible only on the teacher's provided solution, see Section 4). Some own tests could be made private so that they are not shared with peers (we added this feature but retracted it after it created confusion with students, see Section 3.3).

Teacher Monitoring For the teacher to monitor the peer-testing activity and the learning the students go through, the peer-testing platform should offer the ability to track the students' attempts at testing their own solution and their peers' solutions. Such monitoring view could form a testing-based learners log which the teacher would use in assessment. Tracking everything a student has done and putting this in one place could offer a way for the teacher to give feedback on a given coursework (this feature was implemented but is not presented in this paper).

Coursework Artifacts The coursework specification needs to provide interfaces where testing will be conducted. These interfaces could give rise to signature case tests that the teacher provides to ensure that the student solution met the interface required (we have made such teacher provided tests possible, see Section 4).

Tests could be crafted based on input only, the expected output is then produced by a standard solution or oracle that the teacher provides. Students would have the ability to run their own test cases on the oracle solution without having access to its source code (in the current platform, testing is done with unit tests, we have made such teacher oracle implementation part of the coursework setup, see Section 4).

Providing a signature test case and an oracle is to assist the student in crafting solutions and tests. This could be complemented by additional analysis or testing providing automated feedback to the students, but this goes beyond the core purpose of the platform.

Technical Transparency The testing framework of the platform should not hide the technical details from the students: the commands that are run should be displayed; if issues are detected automatically with pre-set verification at compile time or at runtime, the students should be invited to alter their submission rather than having the system pre-processing it (in the current implementation the processing of output is limited to hiding execution paths, command lines are not printed).

Anonymity As our platform is intended for peer feedback, we asked the question *should peer-testing be anonymous?* This would require to hide identifications in submitted files (or to ask the students to do so). The advantages are numerous as it extends to gender anonymity, campus anonymity, and would allow for teacher-crafted submissions and tests to be included as peer submissions. An alternative to asking the students to anonymise their submission is to sanitise the solutions submitted to keep identities anonymous by for instance removing source code comments, but this would hinder code comprehension.

During a peer assessment / automated testing experiment performed by Sitthiworachart and Joy [2004], the students remarked about the use of anonymity. One students response suggested that in a non-anonymous peer assessment exercise, their marking would be influenced if they knew who it was they were marking. Another student said they would decline to take part if the peer assessment was not conducted anonymously. What can be drawn from these comments is that anonymity is clearly very important to those taking part in the peer assessment. But the opinions of anonymity might not correlate with the actual effect it has. To study the effect that anonymity has, as opposed to just opinions about it, a study was conducted by Li [2016]. This study aimed to investigate just how effective anonymity is when it comes to making peer assessment more effective, and whether any negative impact from a lack of anonymity can be mitigated. This quasi-experimental study was conducted with some in-training teachers, and aimed to see which is the most effective method of / using / while conducting a peer assessment exercise: Having assessors and assessees know each others identities, remain anonymous, or know identities while having received training. The training in this case involved watching a video that described some of the stresses and concerns that arise during peer assessment, and various forms of discussion regarding this. The study did not cover the case of being anonymous and getting training. This was because the training was intended as a fallback for when anonymity is not possible. But this does invite the question of just how effective would anonymity be if training were offered as well. The study found: that "anonymity improves student performance in peer assessment", that if anonymity cannot be guaranteed, negative effects arising from this can be offset using training,and that anonymity does not reduce the pressure and tension related with peer assessment.

Currently our platform does not perform any automated anonymisation of the code submitted, instead the students are provided with instructions to remove their name and any other mean of identification from their code before submission or during the online discussions.

3.3 Experiment, Questionnaire and Focus Group Discussion with Students

We conducted an experiment with students studying in year 2 and 3 of the CS department of Heriot-Watt University. The participants were invited to prepare solutions for two small programming exercises (in Python). For each exercise, each student was paired with another student and invited to test their peer's solution. The peer-testing was to be done via the prototype Web platform for the first exercise and via email exchanges for the second one. Following this experiment, the participants were asked to complete a questionnaire and were invited to take part in a focus group discussion in order

to tease out the ways to improve the website further and to gain an insight into their opinions on the peer-testing activity, this latter aspect is discussed in [Grov et al. 2017]. As the programmes are taught in two campuses of the University, namely Edinburgh and Dubai, we had participants from both locations. The way students perceived positively peer-testing as an opportunity for cross-campuses interactions is also discussed in [Grov et al. 2017]. While the participants were well distributed across our campuses (5 in Edinburgh, 6 in Dubai) and in terms of gender (4 identified as female, 7 identified as male), their small number does not allow for quantitative analysis of the feedback. We could however conduct qualitative analysis of the questionnaire answers, and the focus group discussion that followed the experiment. These are discussed below. In general, the comments made by the participants suggest that such website would be welcome in future iterations of CS courses.

Importance of Training Points raised in answers to the question *Was the website behaving correctly? Were there any bugs?* suggested that the participants were put off by the number of failing test cases. Multiple respondents felt that the failing tests limited their ability to provide meaningful feedback. Further investigation into the tests that were submitted to the website, suggested that most of these tests were not failing as a result of website implementation, but were actually related to issues in how participants had written the testing code in python. The way to reduce this issue is to train students to writing test cases and to better explain how the website will run the tests.

The peer-testing platform should first be used in formative lab sessions exercises before being used for an summative coursework to make sure students are familiar with its use.

Peer Group The issue of students not engaging with the peer activity occurred during the experiment: since participants were paired for each exercises, some participants were prevented from taking part since their paired peer did not engage with the exercise.

Organising peer groups of students rather than pairs of students should alleviate this issue, as suggested earlier in Section 2.1. Additionally, peer feedback practices recommend to put in place incentives for students to engage in the peer-testing activity such as to allocate some marks for taking part and for submitting a reflective report after the activity.

Feedback Discussion One issue that appeared in both the survey, and while participants interacted with the website, was that once a feedback comment has been submitted for a test match, it cannot be modified. This caused issues for one participant as they had accidentally clicked the submit button. A solution is to allow feedback to be changed after it has been submitted, although a history of responses would need to be kept. Investigating some of the email-based peer-testing feedback, it looked as though many of the students were providing feedback on their peer's solution as well as on the test cases submitted by their peer's. In the discussion session, a participant suggested to use a 2-way discussion on the website.

A 2-way discussion between pairs of participants (currently implemented, see Section 4) or a forum-style discussion within peer groups would be beneficial in giving more opportunities for feedback.

Usability The evaluation revealed many issues with the usability of the website. The main design of the website and the lack of clear explanation as to how it would function caused some confusion in both how to use the website, and how the various features that were available fit together. This in turn resulted in requiring a lot of effort on the part of the users to get the functionality working right. A number of improvements were carried out since the prototype used in the experiment (contextual help, removing the ability to mark some uploads as *private*, re-design some layouts, mobile-friendly layout).

4 Peer-Testing Platform

The peer-testing platform we are presenting and reporting on in this paper was implemented in Python using the Django Web framework. A web-page[1] dedicated to the peer-testing project provides additional information, and the source code is available on GitHub[2]. The current implementation supports Python (with the `unittest` package for running tests) and Java (with JUnit).

In this section we describe the workflow of using peer-testing with our platform within a coursework assignment.

Peer-Testing Stages The website helps to split up the completion of the coursework into several stages, each with different ability to interact with the website and coursework. These stages, and interactions from the teachers and students involved are detailed in Table 1, and screenshots resulting form some website actions are shown in Figure 1.

In the following we will illustrate these stages of a peer-testing activity with a small exercise requiring the students to implement a sorting algorithm in Python.

Stage 0: Coursework Setup The teachers prepares the materials for the coursework exercise. This would be a document the giving the exercises instructions and learning objectives, in this case implementing the QuickSort algorithm in Python. The teachers could also provide sample solutions and tests.

A sample solution could be an implementation that meets the specification so that student can evaluate the effectiveness of their test cases, we call such sample implementation an oracle. Although students could run their tests on the oracle, its source code will not be

[1] http://www.macs.hw.ac.uk/~mm894/peer-testing
[2] https://github.com/peergramming/peer-testing

accessible to them. Another sample solution could be a template that student could use as a starting point. A sample test could be a simple test case which is used to make sure student solution matches the interfaces of the specification, we call such test a signature test case. Other sample test cases could be provided that student solution are expected to pass for getting certain marks. Once this stage is achieved the students would have access to the coursework information through the coursework's home page, see Figure 1a.

Stage 1: Development & Self-Testing The students are working individually on their implementation. They can upload the source files of their solutions as well as tests they have prepared, see Figure 1b. They can then start running their implementation against the provided test cases or their own tests, they could also run these tests on the oracle, see Figure 1c. Once a run has been requested and performed by the platform, the student could investigate its result as shown in Figure 1d. The results view provides tabs to display the source code of the implementation being tested, the shows, the test case source code, and the output resulting from the running of the test.

Stage 2: Peer-Testing & Feedback When the teachers moves the platform to peer-testing stages, solutions cannot be modified and students are now expected to test the solutions of their peers (each student belongs to a group of peers). In the same way they requested runs of tests on their own implementation, they can now run tests on their peers' solutions, see Figure 1e. The result view now contains a discussion board where the student-tester and the student-developer can enter in a feedback discussion about a run, see Figure 1f.

Stage 3: Teacher Feedback After a deadline announced in advance to the students, the teacher moves the platform to the last stage where no new upload of solution or tests are allowed and where students are expected to write a reflective report of their peer-testing experience. The teacher will be giving feedback and marks on the report and the students' uploads.

5 Related Works

In this section we explore approaches that have been explored to provide peer feedback on programming. We compare these related work with our platform. This review of related work is by no means exhaustive.

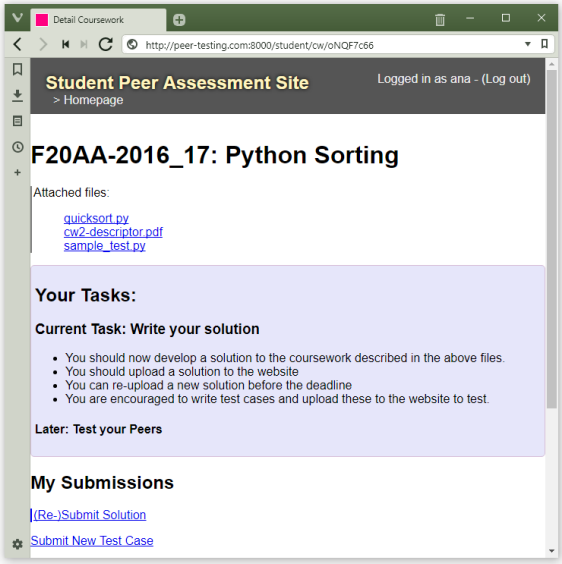
5.1 Testing in Peer Learning Activities

Goldwasser [2002] started a peer testing initiative on a Data Structures course. The main objective of this was to get students taking the course to submit, along with coursework, a test case to run on the program with the aim of finding bugs in other students' programs. The main objective in this case was to focus learning on software testing, and the peer testing aspect was secondary, however the study does hold many parallels with our platform. The submitted coursework solutions would be collected, along with each test case. Then, using an automated system each of the test cases would be run against each of the solutions. The major downside to this cross testing activity being that there is x^2 complexity in terms of the submission count.

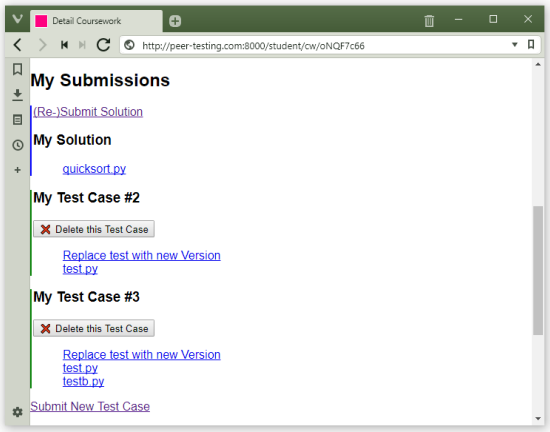
Clark [2004] reports on a 6-year long experiment in having teams of students test their peers' implementations in the context of a software engineering project. The study highlights the positive learning outcomes of such activity to increase the quality of the work, the quality of group collaboration, and in making students understand the necessity of testing. Our platform aims to replicate these results in the context of individual coursework where the management pf peer testing and peer feedback in general is an issue

Reily et al. [2009] presented the system they developed for program peer assessment also called peer code review in a non educational context. As part of the peer assessment, their system required student reviewers to submit test cases results as part of their review as well as an evaluation of the code quality. They structured the peer assessment written comments through a proforma where reviewers are to indicate the outcome of each test case (choosing between *does not compile*, *fatal error/crash*, *incorrect result(s)*, or *correct result(s)*) and are to rate the code quality following criteria (*code formatting*, *conventions*, *documentation*, *interface design*, *modularity*). Similarly, feedback is provided back on the review following criteria (*accuracy*, *helpfulness*, *reviewer knowledge*, *fairness*). Although test cases are the cornerstone of feedback like in our platform, these tests are not run by their system but left to the students to manage, which makes reusing test cases on multiple solutions more tedious to run. The categorisation of test outputs, and feedback seem to be of benefits for structuring the reviews and automating the grading of solutions and reviews. We would consider this approach in future extension of our platform.

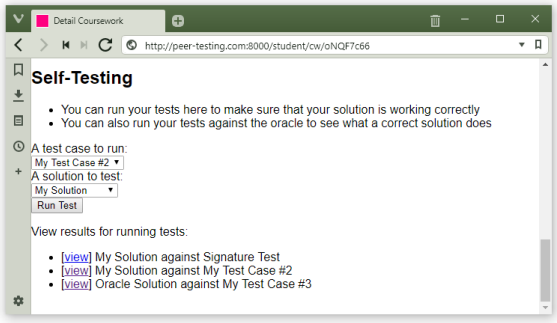
Zeller [2000] describe the Praktomat system they have developed which manages students' peer feedback on programs. The system performs automatic testing of the student submissions. The submission has to pass public tests (provided by the lecturer and available to the students) to be submitted. The system runs additional secret tests (provided by the lecturer but not shared with the students) for assessment by the teacher. Once submitted, students could review each other's program. The students themselves are not required to submit or perform tests. While student considered positively the fairness of testing all submissions with the same set of tests, they initially found the fault-finding system to be fussy.



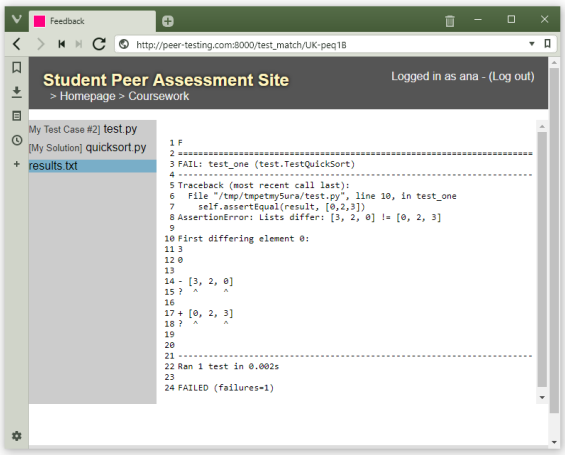
(a) Stage 0: Coursework Setup



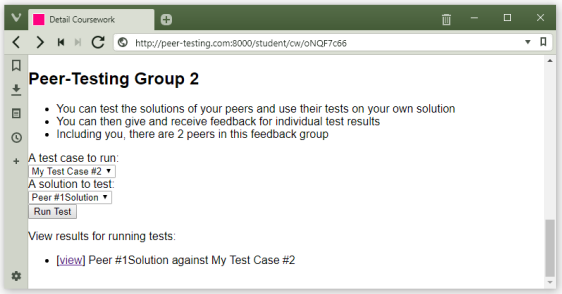
(b) Stage 1: Development & Self-Testing – Uploading solutions and tests



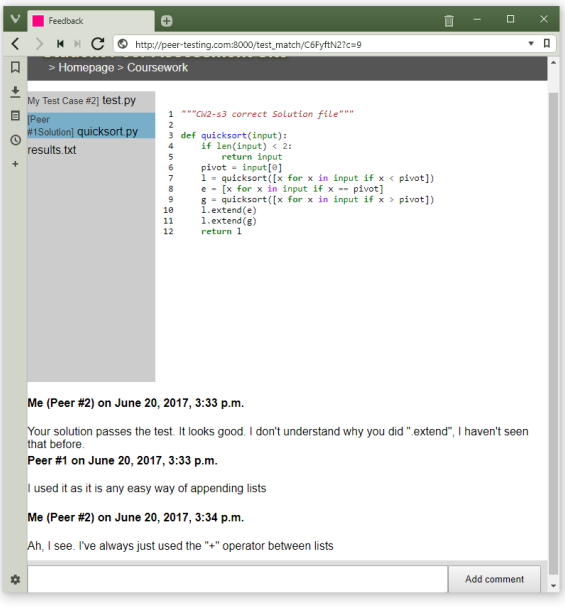
(c) Stage 1: Development & Self-Testing – Running tests



(d) Stage 1: Development & Self-Testing – Viewing test results



(e) Stage 2: Peer-Testing & Feedback – Running tests



(f) Stage 2 Peer-Testing & Feedback – Tests results and feedback discussion

Figure 1. Screenshots of Peer-Testing Stages

Stage		Solutions upload Tests upload Self-testing Peer-testing	Students	Teachers
0	Coursework Setup	✗ ✗ ✗ ✗	Not involved at this stage.	Setting Coursework specification, and program interfaces. Enrolling students and setting up peer groups. Preparing oracle solution and tests.
1	Development & Self-Testing	✓ ✓ ✓ ✗	Develop and upload solutions and tests. Run own tests on own solutions and on oracle solution. Run provided tests on own solutions.	Observing individual progress. Amending peer groups. Adding extra tests or solutions.
2	Peer-Testing & Feedback	✗ ✓ ✓ ✓	Run tests on peers' solutions, review peers' code to send feedback. Receive tests and feedback. Enter discussions with peers.	Monitoring students discussions. Prepare individual feedback.
3	Teacher Feedback	✗ ✗ ✗ ✗	Submit an experience report detailing how they would take on-board feedback they received to improve their program and how they compare their solution with their peers.	Provide group feedback at group level.

Table 1. Peer-testing stages implemented in the Web platform

The table indicates for each stage whether students are allowed to submit artifacts solutions, to submit test cases, to test their own implementation, to view and test their peers solutions. It also describes the expected actions by students and teachers participants.

5.2 Online Peer Learning

A study by Davies [2000] was conducted in a University's communications and networking module using a peer assessment system . This peer assessment system aimed to help students easily provide feedback on reports, in addition to attempting to locate plagiarism. The Computerised Assessment with Plagiarism (CAP) provided an interface to let students read the report of another student. They could then provide feedback on the report and follow references on the report to identify any *copy & paste* plagiarism. The study followed the results students achieved over a sequence of 4 tasks (prepare a report, answer multiple choice test, engage in peer marking of the reports, answer a final multiple choice test). The experiment looked to see if the peer marking was at all helpful in improving the marks of students between the two tests. The results found that the peer marking was very useful for students who had initially performed poorly. Student comments on the experience would suggest that they found the peer assessment process both enjoyable and informative. There is also evidence from this feedback indicating the importance of maintaining anonymity, as some students felt it would be difficult to do the marking non-anonymously. This study also revealed that students learnt a lot from the "repetitive nature of the marking".

To determine whether students can benefit from peer assessment in an online, *technology-enhanced* environment, a research project was conducted by Keppell et al. [2006]. This project involved some university lecturers, who restructured their courses to involve more peer learning, with the aid of a Virtual Learning Environment (VLE). The project took place over three different case studies. The different studies used journal, discussion and file sharing features of the VLE to enhance two courses in fashion design, art education and the design of a new learning website. The evidence collected during these case studies, showed that students found this assessment to be fairer than just against teacher; students appreciated the ongoing peer critique performed through reflective journals; teachers felt the instant nature of feedback was very useful; the support offered by the assisting technology of the VLE encouraged collaboration. The authors of the project also suggest that unless the peer assessment within the VLEs is a marked process (against a students grade), students may be unwilling to participate.

A number of other peer assessment systems exist, see [Luxton-Reilly 2009] for a more substantial survey.

5.3 Automatic Assessment

In the 1990s, the University of Nottingham produced *Ceilidh*, a system that had the intent of automating and improving the assessment process for C Language Coursework. Later improvements were made by Heriot-Watt University adding support for Standard ML (SML) [Foubister et al. 1997], the system was later renamed CourseMarker [Higgins et al. 2003]. This tool was used to guide students through the coursework, and then check the correctness of submissions. Ceilidh offered a skeleton answer to students, which pointed out any special language features that should be used. The submitted solutions are checked for correctness with a mixture of verifying output against a model solution, and checking the style of the code (e.g line count, use of certain function names, etc). In particular for SML, students were asked to provide the types for written functions, as an additional test of understanding. To check if such a system used for automatic assessment of students would be useful, the results of the final exam mark were compared to those marks from previous years where Ceilidh was not used. The study found that there was no real improvement made by Ceilidh, but most importantly there were no detrimental effects. Some positive side effects were observed: the time taken for students to receive feedback on the coursework was markedly decreased, and the tracking of progress offered by the tool allowed teachers to more easily identify any students that were having serious difficulties.

A number of other automated programming assessment systems and approaches exist, see [Ala-Mutka 2005] for a more complete investigation.

5.4 Learning Game and Competition

Tillmann et al. [2013] report on the system Pex4Fun they had designed which automated testing to build a gaming-puzzle. In this system students are expected to attempt to replicate the sample-solution program hidden from them from a empty or faulty implementation. The system uses symbolic execution of the student's version and the secret version to provide feedback to the student. The system was successful as a platform for teaching and learning to program. Pex4Fun later evolved into a new game called Code Hunt [Bishop et al. 2015; Horspool et al. 2015] which combines multiple programming puzzles of the kind Pex4Fun uses. By organising puzzles into difficulty levels and combining these to form a contest, Code Hunt creates an incentive to enter the coding-game. The levels are re-evaluated using players' statistics. Beyond being serious games, an interesting aspect of both these systems is the absence of specification in the coding exercises which makes the participants to reflect on testing results to reverse-engineer the secret program. They also limit the feedback given to a small number of inputs and results not to overwhelm the participant with information which is something our platform currently leave to the peer interaction between students to manage.

Ruef et al. [2016] introduce the programming contest *Build-it, Break-it, Fix-it* that uses the concept of a peer security testing to judge the success of various programming assignments. This success is measured both in terms of general correctness and specifically in the context of security. Although the contest was used in the context of a Massive Open Online Course (MOOC), its aim is specifically targeting security and therefore could not directly be employed for educational programming peer assessment. The contest requires participants to *build* a working solution that matches functional specifications, as well as being as secure as possible. If the solution can be built and the functionality verified by an automated system, then the team can move on to the next stage. Following this, testers will attempt to *break* the security of the solution, and points will be allocated according to a zero-sum-game. The builders can then gain points by *fixing* their solutions. Throughout the contest, there is a central web-based infrastructure that manages running the contest website (with scoreboards, etc.), listening to builder git repositories, and also running and recording test results.

6 Conclusion

We have presented in this paper a Web platform for peer-testing of programming code and the process of its development which was informed by both teachers and students. While the platform is still a work in progress, we will be deploying it this year in our Data Structures and Algorithms course which is simultaneously taught at Heriot-Watt University's Edinburgh and Dubai campuses. We will be collecting feedback on the platform from the students and expect as a result to discover new issues and avenues for improvements.

The platform combines the delivery of coursework assignments within CS programming courses, the self testing of solutions to said coursework, and subsequently the peer-testing that can take place after submission of the coursework is complete.

6.1 Future Works

We are contemplating a number of options to further develop our platform. While we have developed a system from scratch we believe that some features of the platform would be better handled by specialised systems. The platform would therefore benefit from a Git integration for managing students codes as is for instance the system by Ruef et al. [2016], from an integration into VLEs for the management of students submission and grading, and from an integration with existing generic peer feedback or peer assessment systems to manage peer interactions. As discussed in Section 3.3, the feedback discussion board could be extended to include teacher interactions and forum-style discussions. Furthermore, the platform would benefit from integrating peer code

review as done by Trytten [2005] and Hundhausen et al. [2013] and offer a classification of tests and errors as proposed by [Søndergaard and Mulder 2012] and [Reily et al. 2009].

Acknowledgments

This work was in part supported by a Heriot-Watt University and QAA joint funding.

References

Kirsti M. Ala-Mutka. 2005. A Survey of Automated Assessment Approaches for Programming Assignments. *Computer Science Education* 15, 2 (June 2005), 83–102. https://doi.org/10.1080/08993400500150747

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

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

Phil Davies. 2000. Computerized Peer Assessment. *Innovations in Education & Training International* 37, 4 (2000), 346–355. https://doi.org/10.1080/135580000750052955

Nancy Falchikov. 2013. *Improving Assessment Through Student Involvement: Practical Solutions for Aiding Learning in Higher and Further Education*. Routledge.

Sandra P. Foubister, G. J. Michaelson, and Nils Tomes. 1997. Automatic assessment of elementary Standard ML programs using Ceilidh. *Journal of Computer Assisted Learning* 13, 2 (1997), 99–108. https://doi.org/10.1046/j.1365-2729.1997.00012.x

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

Gudmund Grov, Mohammad Hamdan, Smitha S 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 artifacts. In *Horizons in STEM Higher Education Conference: Making Connections and Sharing Pedagogy*. Edinburgh, UK. (presentation in June 2017, paper in preparation).

Colin Higgins, Tarek Hegazy, Pavlos Symeonidis, and Athanasios Tsintsifas. 2003. The CourseMarker CBA System: Improvements over Ceilidh. *Education and Information Technologies* 8, 3 (2003), 287–304. https://doi.org/10.1023/A:1026364126982

R. Nigel Horspool, Judith Bishop, Jonathan de Halleux, and Nikolai Tillmann. 2015. Experience with Constructing Code Hunt Contests. In *Proceedings of the 1st International Workshop on Code Hunt Workshop on Educational Software Engineering (CHESE 2015)*. ACM, New York, NY, USA, 1–4. https://doi.org/10.1145/2792404.2792405

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

Mike Keppell, Eliza Au, Ada Ma, and Christine Chan. 2006. Peer learning and learning-oriented assessment in technology-enhanced environments. *Assessment & Evaluation in Higher Education* 31, 4 (Aug. 2006), 453–464. https://doi.org/10.1080/02602930600679159

Bill Laboon. 2016. *A Friendly Introduction to Software Testing* (1 ed.). CreateSpace Independent Publishing Platform.

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

Lan Li, Xiongyi Liu, and Allen L. Steckelberg. 2010. Assessor or assessee: How student learning improves by giving and receiving peer feedback. *British Journal of Educational Technology* 41, 3 (May 2010), 525–536. https://doi.org/10.1111/j.1467-8535.2009.00968.x

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

Léon McGregor. 2017. Web Platform for Code Peer-Testing. BSc Honours dissertation, Heriot-Watt University. (April 2017).

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

Andrew Ruef, Michael W. Hicks, James Parker, Dave Levin, Michelle L. Mazurek, and Piotr Mardziel. 2016. Build It, Break It, Fix It: Contesting Secure Development. In *ACM SIGSAC Conference on Computer and Communications Security (CCS)*. Vienna, Austria, 690–703. https://doi.org/10.1145/2976749.2978382

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

Jirarat Sitthiworachart and Mike Joy. 2004. Effective peer assessment for learning computer programming. ACM Press, 122. https://doi.org/10.1145/1007996.1008030

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

Harald Søndergaard and Raoul A. Mulder. 2012. Collaborative Learning through Formative Peer Review: Pedagogy, Programs and Potential. *Computer Science Education* 22, 4 (Dec. 2012), 343–367. https://doi.org/10.1080/08993408.2012.728041

Nikolai Tillmann, Jonathan de Halleux, Tao Xie, Sumit Gulwani, and Judith Bishop. 2013. Teaching and Learning Programming and Software Engineering via Interactive Gaming. In *Proceedings of the 2013 International Conference on Software Engineering (ICSE '13)*. IEEE Press, Piscataway, NJ, USA, 1117–1126. https://doi.org/10.1109/ICSE.2013.6606662

Keith J. Topping. 2009. Peer Assessment. *Theory Into Practice* 48, 1 (2009), 20–27. https://doi.org/10.1080/00405840802577569

Deborah A. Trytten. 2005. A Design for Team Peer Code Review. In *Proceedings of the 36th SIGCSE Technical Symposium on Computer Science Education (SIGCSE '05)*. ACM, New York, NY, USA, 455–459. https://doi.org/10.1145/1047344.1047492

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