CHALLENGES – TOWARDS CONTINUOUS PEER ASSESSMENT IN UNDERGRADUATE PROGRAMMING CLASSES

Manfred Meyer
Department or Mechanical Engineering,
Westfälische Hochschule - Westphalian University of Applied Sciences (Germany)

Abstract

This paper presents a pragmatic approach for stepwise introduction of peer assessment elements in undergraduate programming classes, discusses some lessons learned so far and directions for further work. Students are invited to challenge their peers with their own programming exercises to be submitted through Moodle and evaluated by other students according to a predefined rubric and supervised by teaching assistants. Preliminary results show an increased activation and motivation of students leading to a better performance in the final programming exams.

Keywords: *Continuous assessment, peer assessment, formative assessment, competency-oriented exams.*

---

1. Introduction and overview

Peer assessment has become quite popular as a means to empower students to take responsibility for and manage their own learning while also learning to assess and provide constructive feedback to peers (Cornell University, 2023). Besides enhancing the effectiveness of learning itself ("assessment for learning"), peer assessment can also serve for activating students as instructional resources for each other.

Computer programming can be regarded as an essential skill for the 21st century. Introductory programming courses are thus popular in higher education as part of the foundations of an information technology-related curriculum (Ng, 2012). However, as computer programming is a complex intellectual activity, programming classes are regarded as difficult resulting in often high dropout rates. Therefore, many researchers and practitioners have proposed various methodologies and tools to help students learn computer programming.

In this paper we first discuss the background of undergraduate computer programming classes in general and in our specific case, review some related work on using peer assessment in this context before we present our approach in detail concluded by a preliminary evaluation and a discussion of some directions for future work.

2. Background

The difficulties involved in learning how to program have various aspects ranging from the syntax of programming languages that have not been designed for novices to the need for abstraction when developing algorithms and the use of state-of-the-art tools for efficient and effective programming, a competency required in professional software development but rarely taught at university.

Therefore, more than a decade ago, we redesigned our first-year programming courses (GDP1 and GDP2 at our university, internationally often referred to as CS100/101 courses) in a stringent competency-oriented manner. We now strictly focus on the key competencies of (1) understanding a problem statement, (2) modelling the required data and deriving an algorithm to solve it, and (3) implementing and testing the resulting Java program using Eclipse as state-of-the-art Integrated Development Environment (IDE). Additionally, we introduced the social learning platform *Perusall* (Miller, Lukoff, King, & Mazur, 2018) in order to get students better prepared for the lectures, enforce social interaction between peers and leave more of the precious contact time for in-depth discussion of "hard topics" (as identified by Perusall's "Confusion Report") and the use of "Just-in-Time Teaching (JiTT)" methods (Camp, Middendorf, & Subiño Sullivan, 2010) like Peer Instruction (Mazur, 1997) or Flipped Classroom (Bergmann & Sams, 2012).

Consequently, in order to ensure the 'constructive alignment' of learning and assessment according to (Biggs & Tang, 2011), students are informed from the very beginning of their classes, that it is exactly this competency that is getting tested in the final exams. There are no questions like naming all

primitive data types in Java or transforming a for-loop in an equivalent while-loop, but knowledge about data types or loop-statements is nevertheless essential to solve the given programming exercise.

For the practical classes in the lab, we asked the students to work on the programming exercises in small teams of around four students sharing one table. We also motivated them to continue working in teams off campus and provided additional exercises to take home for training purposes between classes and over the weekends.

Having recognized that those students performed better in the final programming exams who have been learning/training in teams continuously asking for some more exercises or even defining their own training cases, we started investigating how to enforce this type of "peer challenging" more formally and facilitate constructive peer feedback. This we later developed further into a peer assessment approach rewarding bonus points for the final exam in order to activate students and motivate them to take part in this feedback culture.

3. Related work

While peer assessment has been successfully employed in a variety of academic disciplines and its use in the classroom in order to increase student engagement by actively involving them in the assessment process has been practiced and researched for decades, there is not much work yet on using peer assessment in programming classes.

Sitthiworachart and Joy (2004) already proposed a web-based peer assessment system in the context of an undergraduate computer programming course. Dolezal, Motschnig, and Pucher (2018) analyze the applicability of peer assessment to smaller exercises at secondary school level and makes recommendations for its use in computer science courses.

Alkhalifa and Devlin (2021) focus on the crucial role that students play in peer assessment, especially on programming students' perspectives on such practices. They also discuss the students' expectations and critical issues during development of the peer assessment system.

While in our approach we focus on feedback being given and assessment made by students themselves, Srikant and Aggarwal (2013) discuss the automatic evaluation of computer programs for their potential for large-scale impact. They present a machine learning framework to automatically grade computer programs and claim that it provides a better grading than a simple test-case-pass based grading which we also use in our approach but only as one of many aspects in the grading rubric.

4. Our approach

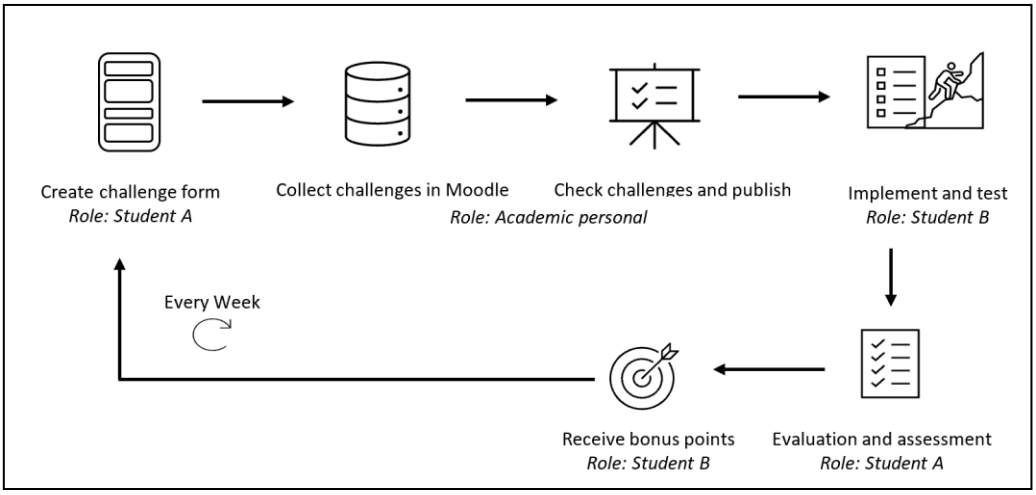
In our programming classes, we introduced weekly 'challenges': programming exercises submitted by students to be solved by their peers. They are currently used as continuous formative assessment throughout the lecturing period by which students can collect 'bonus points' (for both submitting and solving challenges) which can get credited up to 20% towards the summative assessment, thus reducing stress at the final exam.

As our programming courses have been designed as entirely competency-oriented and constructively aligned classes and exams (exam notebooks provided with IDE and state-of-the-art programming tools), continuously solving challenges provides a perfect training for the final exam. Moreover, by designing and submitting 'good' challenges, students demonstrate deeper understanding of the topic while balancing its complexity: a 'good' challenge is challenging enough to get solved by many but probably not all students.

The current implementation is integrated into the Moodle LMS where students can submit challenges through a form basically asking for the exercise statement (text) plus Java code template (optional) and test cases (input and expected output). Challenges are then checked by teaching assistants (TAs) for validity and understandability, and selected challenges get presented to all students on Moodle. With the use of the CodeRunner plugin, the provided test cases help students check the functionality of their solution while design aspects etc. are being assessed manually – by the authors of the respective challenge currently still being supervised by TAs for evaluation and ensuring assessment quality and fairness.

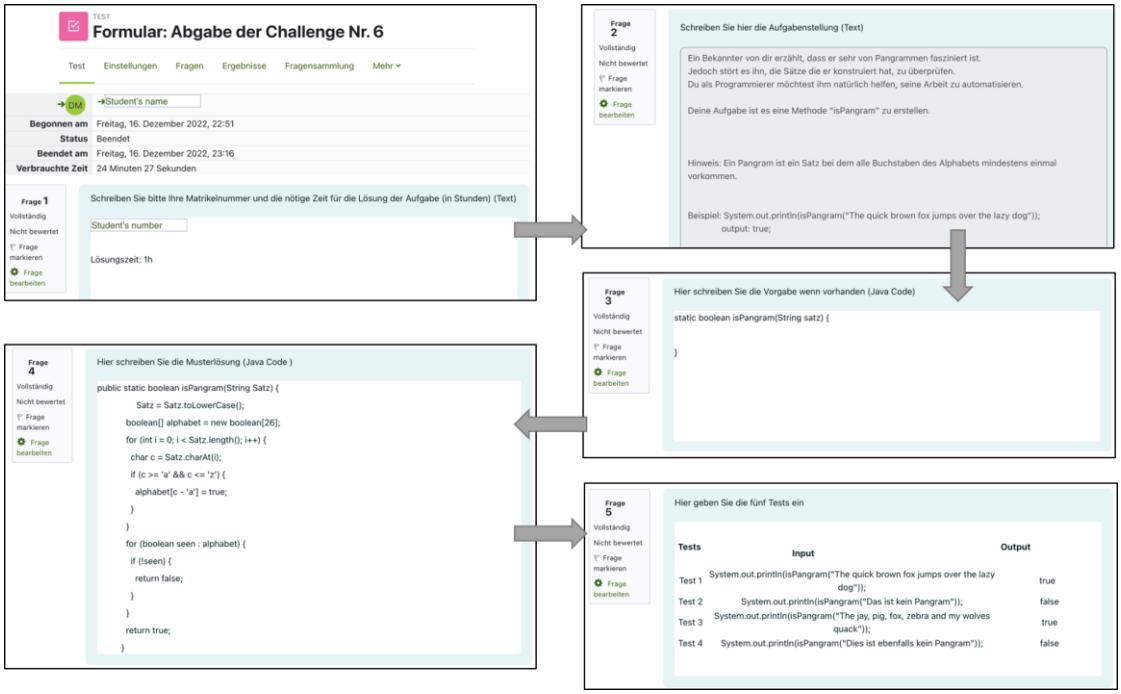
While our approach extends usual peer assessment by also asking students to design the assessment ('challenge') itself, its introduction in first programming classes also revealed several lessons learned: from student's participation (acceptance), the need for clear rubrics and prior assessment training to technical issues with Moodle integration. Moreover, the ultimate goal of entirely substituting the final exam by continuous assessment throughout the lecturing period requires substantial changes to the examination regulations. However, preliminary feedback from students shows that the majority (but not all) like this approach both for continuous training for the exam and also for improving their marks by bonus points collected through challenges.

Figure 1. The Peer Challenges Process.



Students are invited to submit so-called "Challenges" in Moodle through a predefined form using the Test feature of Moodle. It comes with five questions ("Frage") asking for the student's ID and the estimated time needed for solving the challenge (Frage 1) followed by the detailed introduction to the given exercise and explanation what to do (as plain text, "Frage 2"), a Java code template ("Frage 3"), a sample solution (Java code, "Frage 4") and concluded by a list of test cases, each consisting of a Java statement and its expected output ("Frage 5").

Figure 2. Submitting a new Challenge (student's view).

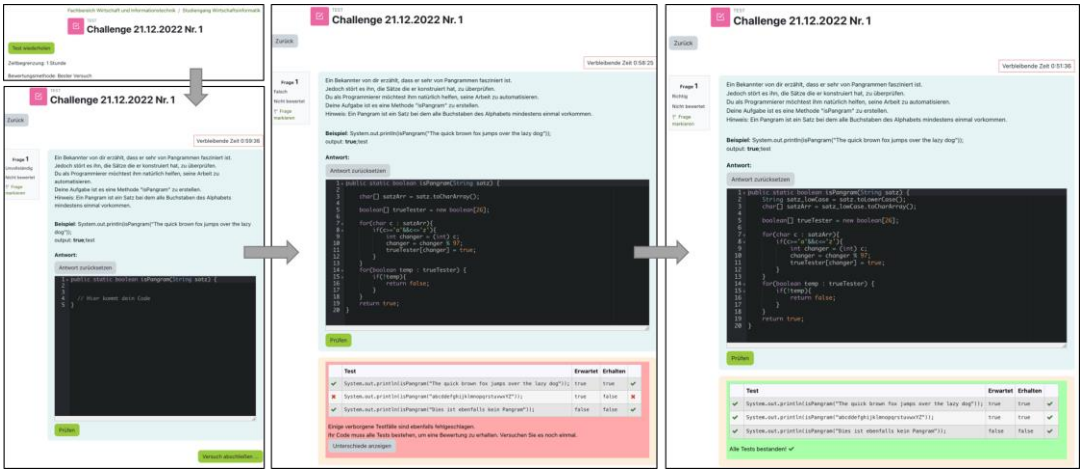


Once a challenge is submitted, it gets checked by TAs for correctness and completeness, understandability, appropriate test cases, difficulty and time estimate to solve it. If a challenge passes these checks, it gets published on Moodle, otherwise the student who submitted the challenge gets detailed feedback why it didn't get published and how to improve for resubmission. Additionally, authors of challenges being published receive up to three bonus points towards the final programming exam.

All accepted challenges - plus some ready-prepared challenges in case there are not enough submissions from students – get published on Moodle on a weekly basis giving students one week to work on them. While students can use any external IDE to develop a solution (Java program), they can also directly use the built-in editor that comes with the CodeRunner plugin for Moodle where students can iteratively work on the code until all visible test cases are being passed and the solution can be

formally submitted. There is no time limit imposed and students may also work on challenges in teams, however solutions can only be submitted by individual students.

Figure 3. Solving a Challenge (student's view).



In the third phase, all submitted solutions to each challenge need to get evaluated. This is done also by students, normally by those who submitted the challenge (author). However, in rare cases students submitted a challenge but did not feel comfortable with also doing the assessment of their peers. In those rare cases, other students are invited to assess.

In order to ensure a fair assessment, a detailed rubric has been developed and all students doing assessments have been instructed beforehand how to apply this rubric to a given student's submission. However, especially in the beginning until students have developed some routine in using the rubric, they get supported by TAs. Table 1 shows the main criteria being addressed by the rubric. For each criterion, points are associated that get awarded to the respective student depending to what extent the criterion is being met.

Table 1. Criteria from the Peer Assessment Rubric.

• correctness (all tests being passed including some additional tests provided with each challenge which are not visible to the students when submitting their solution) • proper documentation of classes and methods (using Javadoc), • compliance with coding standards (naming, structure, indentation), • simplicity and readability (keep it simple!), • modularity (code structure, use of classes and methods), • robustness (dealing with incorrect input, use of exception handling) • efficiency and scalability of the underlying algorithm, if applicable

Thus, students can collect bonus points both by submitting challenges that get selected for publication on Moodle as well as by solving a given challenge and submitting their solution also through Moodle. However, the overall number of bonus points that can be credited towards the final programming exam is limited to 20% in total by our university's examination regulations.

5. Evaluation and future work

The peer assessment approach presented in this paper has been applied to first-year undergraduate programming classes (Introduction to Programming 1 and 2, GDP1 & GDP2) within the Business Informatics program at our university (Bocholt campus). At the beginning, we first introduced challenges developed by Tas as additional training exercises. This helped testing the implementation in Moodle using the CodeRunner plugin and also activating students and getting them familiar with the process of submitting solutions through Moodle. The awarded bonus points also helped to motivate students to take part in this experiment. Around mid-term we then started inviting students to also submit their own challenges once a sufficient number of students worked on the given challenges quite regularly.

As expected, students showed to be much more hesitant when it came to developing and designing their own challenges, so it took some weeks and also support from Tas until we received

challenges quite regularly such that most of the challenges presented to the students came from their peers. But still student submissions got evaluated and bonus points assigned by Tas.

It appeared that students didn't feel very comfortable in being involved in the evaluation process and thus it needed some time and clarification of the evaluation criteria and standards. Only towards the end of the term, we were able to complete our peer assessment approach by also having the students themselves doing the evaluation of their peers' submissions and giving constructive feedback.

However, after having overcome this restraint, we now observe a small but constant stream of challenges being submitted. Students have now accepted our approach as an additional means for training for the exams while also earning some bonus points already. As there are up to three times more bonus points awarded to submitting challenges and doing the assessment afterwards than for just solving a given challenge, more and more students do also participate in this core part of our approach focusing not only on the programming training and learning aspect but also on developing analytical and communication competencies when it comes to give constructive feedback to their peers.

Summarizing, the informally gathered feedback from the students is now quite positive and our challenges approach is being accepted by most of the students while some students still hesitate to participate. However, a more formal validation of our approach still needs to be done after the end of the first year. Based on the results we will then derive conclusions on how to improve the process and its implementation in the future.

Acknowledgements

The work presented here is part of a Fellowship on Digital Examinations which the author got awarded by the initiative "Prüfung hoch 3" funded by the Stifterverband through the Friedrich-Alexander-University Erlangen-Nürnberg (project ID: H110 5114 5132 36447). The implementation in Moodle has been developed by Sihem Ould Mohand who also prepared the graphics for this paper.

References

Alkhalifa, A., & Devlin, M. (2021). Student Perspectives of Peer Assessment in Programming Courses. In *Proceedings of the 2021 Conference on United Kingdom & Ireland Computing Education Research (UKICER '21)*. 8, 1-7. New York: Association for Computing Machinery.

Bergmann, J., & Sams, A. (2012). *Flip Your Classroom*. Washington, D.C.: International Society for Technology in Education.

Biggs, J., & Tang, C. (2011). *Teaching for Quality Learning at University*. Maidenhead: The Society for Research into Higher Education and Open University Press (McGraw Hill Education).

Camp, M. E, Middendorf, J., & Subiño Sullivan, C. (2010). Using just-in-time teaching to motivate student learning. In S. Simkins, & M. H. Maier (Eds.), *Just-in-time teaching: Across the disciplines, across the academy* (pp. 25-38). Sterling, VA: Stylus Publishing.

Cornell University, Center for Teaching Innovation (2023). *Peer assessment*. Retrieved March 20, 2023, from https://teaching.cornell.edu/teaching-resources/assessment-evaluation/peer-assessment

Dolezal, D., Motschnig, R., & Pucher, R. (2018). Peer Review as a Tool for Person-Centered Learning: Computer Science Education at Secondary School Level. In M. Auer, D. Guralnick, & I. Simonics (Eds), *Teaching and Learning in a Digital World*. *Advances in Intelligent Systems and Computing*, 715. Springer.

Mazur, E. (1997). *Peer Instruction: A User's Manual*. Prentice Hall.

Miller, K., Lukoff, B., King, G., & Mazur, E. (2018). Use of a Social Annotation Platform for Pre-Class Reading Assignments in a Flipped Introductory Physics Class. *Frontiers in Education, 3*.

Ng, W. (2012). The Impact of Peer Assessment and Feedback Strategy in Learning Computer Programming in Higher Education. *Issues in Informing Science and Information Technology, 9*.

Sitthiworachart, J., & Joy, M. (2004) Effective peer assessment for learning computer programming. In *Proceedings of the 9th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education* (pp. 122-126).

Srikant, S., & Aggarwal, V. (2013). Automatic Grading of Computer Programs: A Machine Learning Approach. In *Proceedings of the 12th International Conference on Machine Learning and Applications (ICMLA)* (pp. 85-92).