Atelier – Tutor Moderated Comments in Programming Education

Ansgar Fehnker(✉), Angelika Mader, and Arthur Rump

University of Twente, P.O Box 217, 7500 AE Enschede, The Netherlands
ansgar.fehnker@utwente.nl

Abstract. In the programming course of our engineering design degree tutorials are the focal point of learning. This is especially so since we employ a tinkering based educational approach, in which students explore, from the very beginning, the material by self-defined projects. The assignment defines ingredients to use and sets expectations, but students are free to set their own design goals. In this setting tutorials are an important place of feedback and learning, and we developed an online platform that supports tutors during tutorials. This paper reports on the educational philosophy and underpinnings, and results from applying the tool in two first-year courses.

Keywords: Novice programmers · Online platform · Tutorials · Semi-automated feedback · Community of practice

1 Introduction

This paper presents *Atelier*, an online platform that supports tutoring in programming courses, emphasising collaboration and sharing[1]. It is built for the *Community of Practice* [1] of students, tutors, and lecturers involved in teaching programming, where personal feedback is a core element. *Atelier* is intended to support, but not replace personal tutoring.

The platform has been developed in the context of our bachelor programme *Creative Technology* (*CreaTe*), which is a multidisciplinary programme with a base in computer science and electrical engineering, a strong focus on design, and which includes elements of entrepreneurship. The programming courses of *CreaTe* require students to use concepts that were covered in the course, but they are free to define their own projects from the very beginning. We refer to this approach as *Tinkering* [5]. The student fully owns the problem; there is no example solution that students can work towards or that tutors can refer to.

In this setup, the focal point of learning programming is the tutorial, where students work on their projects, supported by a team of tutors and lecturers. Accordingly, individual feedback is a key element in this teaching approach. The

[1] The Atelier project is supported by SURF as part of its 2018 call on Open and Online Education.

T. De Laet et al. (Eds.): EC-TEL 2021, LNCS 12884, pp. 379–383, 2021.
https://doi.org/10.1007/978-3-030-86436-1_39

Atelier platform provides tutors with automated feedback that is initially only visible to the tutors; feedback they can share and discuss with the student if they see it fit.

Tools that automatically provide feedback have been around since at least the 1970s. Keuning analysed 101 tools and found that the majority look at *knowledge of mistakes*, and there is less attention for the quality of the programs [4]. They found that testing is the most popular technique. Douce, Livingstone and Orwell [2] describe several generations of these tools, which require well-defined exercises with supplied test cases to function correctly. This is a very different setting from ours, as we mainly use open exercises. Keuning studied the use of static analysis tools similar to ours and found that novices often to not fix issues that such tools report, especially problems with design [4]. Keuning theorises that students may simply not know how to fix their code.

Our approach differs from many automated tools in literature with regards to two important aspects. We consciously chose not to use automated tools to replace tutor-student interaction, and we also do not use them for grading. They are used to assist tutors, during tutorials, to provide feedback that is more transparent and consistent. This gives rise to the research question whether warnings that are given to a student by a tool, differ in effectiveness from automated warnings that a tutor shares and follows up.

2 The *Atelier* platform

The aim of *Atelier* is to support the feedback process on student-defined projects in *CreaTe*. Tool support is primarily aimed at helping the tutor. *Atelier* uses two tools, *Zita* to highlight potential programming issues [3], and *Apollo* to estimate whether a student achieves certain learning outcomes [6]. Importantly, both tools are not included for marking, or to substitute tutor feedback, but are meant to aid the tutor.

Setting. The platform *Atelier* was developed for use during programming tutorials when students work on exercises that relate to topics covered by lectures. In line with the *tinkering* approach, students have to incorporate what they have learned in a self-defined project. The tutor should give the student feedback on their code verbally, and the online platform should not substitute this process, but complement it.

Usage Scenario. The primary usage scenario is illustrated in Fig. 1. The student shared the program with a tutor, e.g. via a QR code. During the exchange, the tutor can make notes and comment on the program or individual lines of code. The tutor is also presented with automated *Zita* comments which are initially only visible to the tutor. In this case, the tutor decided to make the comment visible and further elaborate on the warning. The student was able to reply, and the tutor involved another tutor in the discussion.

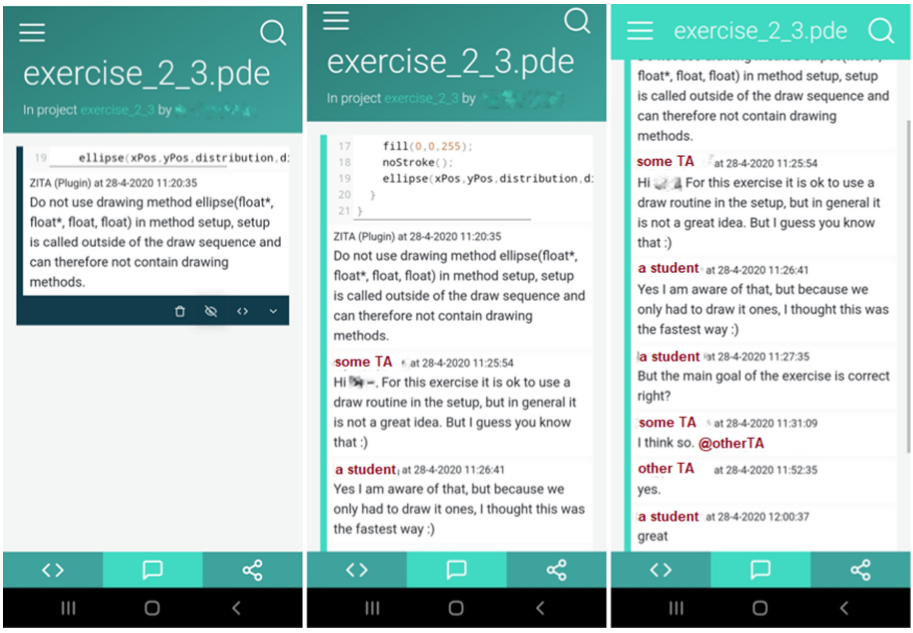

Fig. 1. An exchange from a tutorial, illustrating the main usage scenario. Names have been replaced by generic labels "`a student`", "`some TA`", and "`other TA`".

Implementation. *Atelier* is available as an open-source project on GitHub[2]. The two plugins we developed, *Zita* and *Apollo*, are available under the same GitHub organisation. These projects are not part of the main program, because they are specific to Processing and to our courses.

3 Experience and Observations

Context. The first year of the *CreaTe* bachelor is organised in four modules. A module is an integrated study unit that has several components; for some of these modules this includes programming. *Atelier* was first deployed in module 4 of the first year, which includes an algorithms course.

The course started about a month into the first nationwide lockdown in the Netherlands in response to the COVID19 pandemic. To maintain the spirit of traditional tutorials we chose a synchronous form of teaching, with one main conference, a queuing system for help requests, and breakout rooms.

We used a similar setup for module 1 in the next academic year, for a new cohort of students. This module includes an introductory programming course, with 5 weeks of tutorials and lectures that cover the basics of programming, like variables, decisions and loops, up to objects, classes and arrays.

[2] See https://github.com/creativeprogrammingatelier/atelier.

Usage. Over the two modules we had 211 students and 43 tutors and lecturers use *Atelier*. Students shared in total 809 programs. Users could indicate whether we could use their data for research. This was permitted by 128 students and 33 tutors and lecturers. This left 499 student submissions for analysis, which are used in the remainder. Figure 2 shows that in the research data set, less than 33% of the students submitted 4 or more programs.

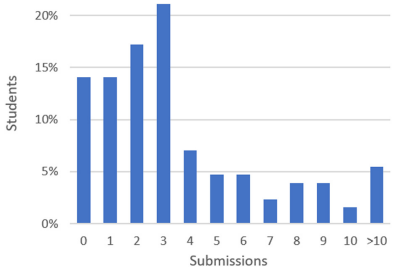

Fig. 2. Histogram of the number of submission per student.

Evaluation. The tools *Zita* and *Apollo* generated 3864 comments, of which 312 were made visible to students. This is a low percentage, and tutors indicated that this is in part because of the repetitive nature of *Zita* comments. For example, one submission received 95 warnings of the same type – a naming convention – but only 2 of them were made visible.

An interesting observation could be made with respect to the effectiveness of sharing *Zita* comments. We distinguish between comments that were shared with the student, and comments that were shared **and** followed up by an additional comment. To measure the effectiveness we define for each submission a window of future submissions and checked whether the same of the 35 *Zita* rules issued another warning within the window.

Figure 3 depicts the results for window sizes up to 5. The left figure shows, e.g., that 54% of the students avoids repeating a mistake relating to a shared warning in the next submission (window size 1). Unfortunately, this share decreases as the window size increases. Five submissions later 78% repeat the same mistake. Note, that the total numbers decline with increasing window size, since there will be fewer submissions by the same student for larger window sizes.

The right figure show that *Zita* comments that were followed up were somewhat more effective, even though that effect also waned with an increasing window size. Note, that the same mistake may be repeated for various reasons, for example because a student finds different ways to violate the same rule.

Threats to Validity. The module 4 course had only 8 weeks of lectures and tutorials, while the module 1 course had only 5 weeks of tutorials, for a different cohort. This means that we could only measure the use of a newly developed tool for a short period. The effect of feedback may change if students are exposed to it for a longer time, positively because of consistent messaging, or negatively because they get used to it.

When considering the effectiveness of tutor feedback, one also has to keep in mind that the student knows the person, and also knows that this person may assess them in the future. This is both a threat to the validity, but also a strength because it introduces a personal aspect into the process.

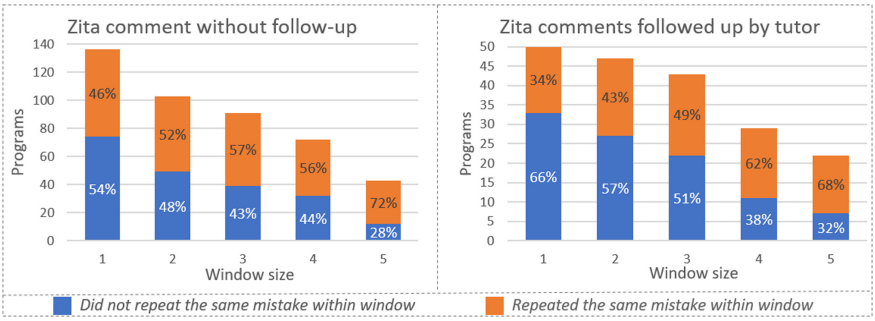

Fig. 3. Number and share of repeated mistakes, depending on whether a *Zita* comment was followed up or not.

4 Conclusions

For a bachelor programme which combines design with engineering approaches, and which has a very diverse student population, we use a teaching method for programming that emphasises creativity, ownership and individual solutions. Given these individual learning paths, providing good feedback is one of the key ingredients of the approach. This paper introduced the platform *Atelier* developed for this purpose, integrating the tool *Zita* for automated feedback. Tutors can give feedback on a program or on certain lines of code, and use automated feedback as a starting point for further discussion, which proved to be somewhat effective.

Currently, *Atelier* is still limited to working with projects created using Processing. This limitation will stay in place while we work to improve the platform, but we are planning to remove these limitations and enable *Atelier* to be used with other programming languages at a later stage.

References

1. Coenders, M.: De Canon van het Leren, chap. Community of Practice - Etienne Wenger, pp. 126–136. Vakmedianet Management B.V. (2012)
2. Douce, C., Livingstone, D., Orwell, J.: Automatic test-based assessment of programming. J. Educ. Resour. Comput. **5**(3), 4-es (2005). https://doi.org/10.1145/1163405.1163409
3. Fehnker, A., de Man, R.: Detecting and addressing design smells in novice Processing programs. In: McLaren, B.M., Reilly, R., Zvacek, S., Uhomoibhi, J. (eds.) CSEDU 2018. CCIS, vol. 1022, pp. 507–531. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-21151-6_24
4. Keuning, H.: Automated feedback for learning code refactoring. Ph.D. thesis, Open Universiteit (2020)
5. Mader, A., Fehnker, A., Dertien, E.: Tinkering in informatics as teaching method. In: CSEDU 2020. Scitepress (2020). https://doi.org/10.5220/0009467304500457
6. Rump, A., Fehnker, A., Mader, A.: Automated assessment of learning objectives in programming assignments. In: Cristea, A.I., Troussas, C. (eds.) ITS 2021. LNCS, vol. 12677, pp. 299–309. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-80421-3_33