Promoting Early Engagement with Programming Assignments Using Scheduled Automated Feedback

Paul Denny
University of Auckland
Auckland, New Zealand
paul@cs.auckland.ac.nz

Jacqueline Whalley
Auckland University of Technology
Auckland, New Zealand
jwhalley@aut.ac.nz

Juho Leinonen
University of Helsinki
Helsinki, Finland
juho.leinonen@helsinki.fi

ABSTRACT

Programming assignments are a common form of assessment in introductory courses and often require substantial work to complete. Students must therefore plan and manage their time carefully, especially leading up to published deadlines. Although time management is an important metacognitive skill that students must develop, it is rarely taught explicitly. Prior research has explored various approaches for reducing procrastination and other unproductive behaviours in students, but these are often ineffective or impractical in large courses. In this work, we investigate a scalable intervention that incentivizes students to begin work early. We provide automatically generated feedback to students who submit their work-in-progress prior to two fixed deadlines scheduled earlier than the final deadline for the assignment. Although voluntary, we find that many students welcome this early feedback and improve the quality of their work across each iteration. Especially for at-risk students, who have failed an earlier module in the course, engaging with the early feedback opportunities results in significantly better work at the time of final submission.

CCS CONCEPTS

• **Social and professional topics** → *Computing education.*

KEYWORDS

early feedback, deadlines, self-regulation, novice programmers, automated feedback, time management, procrastination, assessment, at-risk students

ACM Reference Format:
Paul Denny, Jacqueline Whalley, and Juho Leinonen. 2021. Promoting Early Engagement with Programming Assignments Using Scheduled Automated Feedback. In *Australasian Computing Education Conference (ACE '21), February 2–4, 2021, Virtual, SA, Australia.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3441636.3442309

1 INTRODUCTION

Programming assignments often require a substantial amount of work from students, and this may be spread over a number of days or weeks leading up to a deadline. Time management is therefore a very important metacognitive skill for students to develop. In the programming education literature, a range of activities aimed at supporting metacognitive and self-regulatory skills have specifically focused on time management [31]. Despite this, it is known that many students do not manage their time well and will delay starting work on an assignment if the deadline is not imminent. This is a serious problem as it is often difficult for students to judge how much time it will take them to complete an assignment, particularly if they need to seek help. There is also clear evidence that starting work early, and making incremental progress, is strongly correlated with successful outcomes [9, 25].

As enrolments in programming courses have grown, there has been an increasing reliance on automated marking for reasons of efficiency. Automated assessment tools, which provide students immediate feedback on code they have written, are often used for small programming tasks [26]. Providing automatically generated feedback of program correctness for larger programming assignments is also possible, and can help students to fix errors or improve their submissions prior to a final submission deadline. Such formative re-submissions allow students to learn from their mistakes, reduce careless errors, and can reduce the anxiety associated with the act of submitting an assignment [28]. However, allowing students to access such formative mechanisms at any time does not address issues around time management or procrastination.

In this research, we investigate the use of an optional feedback and re-submission process for an end of course programming assignment in an introductory course. The feedback that students receive is automatically generated enabling this approach to work at scale. To receive feedback on their work, students submit what they have completed prior to a scheduled "early feedback" deadline. The scheduling of these deadlines is designed to encourage students to start their work early, such that the more progress they make, the more useful the feedback they receive. The nature of the feedback is minimal, essentially revealing only the number of functional tests, across a range of independent tasks, that the submitted code passes. This limited feedback still helps students to identify the tasks to which they should allocate their available time, but by not revealing details about the errors, students are expected to review their own code carefully.

In this work, we address the following research questions:

RQ1. To what extent do students take advantage of scheduled, but optional, early feedback opportunities?
RQ2. What relationship exists between receiving early feedback and subsequent assignment performance?
RQ3. Do at-risk students, who failed a prior module, make use of the early feedback opportunities and if so, how does this impact their performance?

ACE '21, February 2–4, 2021, Virtual, SA, Australia

ACM ISBN 978-1-4503-8976-1/21/02…$15.00
https://doi.org/10.1145/3441636.3442309

2 RELATED WORK

It is generally accepted that providing learners with useful, actionable feedback is critical for their progression and engagement. In his best-selling book on teaching in higher education, Ramsden provides a section on the effect of feedback on performance [32, p. 1930]. He notes that for feedback to be useful it needs to be timely and detailed. This presents unique challenges in the computer science classroom. As enrolments increase, providing timely feedback on a large scale can be difficult without the use of automation. Moreover, novice programmers may lack the knowledge, language (i.e. technical terminology) and experience needed to interpret detailed feedback. Ramsden also noted that when feedback is delayed, students do not have sufficient time to learn from it and adjust their behaviors. It is reasonable therefore to expect that providing simple feedback, regularly throughout a programming assignment, should lead to improved learning and better outcomes. But what kind of feedback is appropriate for novice programmers, and can it be scheduled to promote positive time management behaviors and discourage procrastination?

2.1 Feedback and Learning to Program

The earliest, and most common, form of feedback students receive on their programming is from a compiler or interpreter and relates to syntax. The error messages provided by such tools are immediate, but often unhelpful because they are cryptic and difficult for novice programmers to decipher. As a result, there has been a recent surge in interest focused on improving the quality and usefulness of such error messages [2, 11].

One scalable solution for providing feedback on program correctness is to release unit tests or testing code directly to students. This approach has been found to afford some benefits, as the feedback students receive enables them to write fully compliant and functionally correct code, helps them focus on code quality, and increases their confidence in their programming ability [4, 5]. But for many students unit tests are complicated and unhelpful. In one study, novice programmers were given unit test code to help them during programming labs and take-home assignments [37]. At the end of the course, students were surveyed and although they all reported using the provided unit test code, some students found it hard to understand and read, and the resulting error messages difficult to interpret. This suggests that simpler forms of feedback may be more appropriate for novices.

Other research has looked at the usefulness of software metrics [3, 27] as a feedback mechanism. Work in this area also suggests that students find it difficult to interpret metric-based feedback [27], and has lead researchers to explore a variety of feedback formats [24]. Providing convenient and actionable feedback, which is simple for students to understand, remains an ongoing challenge of great interest to computing educators [8].

2.2 Automated Feedback

Automated feedback systems, many of which provide a web-based interface through which code can be uploaded or submitted, provide convenience for students and the ability to collect fine-grained process data for instructors and educational researchers [21, 33]. A wide variety of such systems exist, ranging from practice tools that provide automatic feedback on small fragments of student-written code [10, 13, 30], to more sophisticated frameworks designed to automatically grade large multi-file assignments [14].

A 2010 systematic review of the literature on automated programming assessment tools reported that many systems produce feedback based around test cases that are either automatically generated or taken from instructor-provided test suites [20]. In general, automated feedback is viewed positively by programming students and influences them to put more effort into their work [16]. A review of a range of educational interventions in computer science classrooms shows significant evidence for the benefits of automated feedback [35]. In 2018, Keuning et al. published a systematic literature review on automated feedback for programming exercises [26]. A total of 101 tools were reviewed and classified according to the type of feedback they provide. The authors found that contemporary tools tend to focus more on test failures and compiler errors than their earlier counterparts which focused mainly on feedback related to solution errors. Very few tools provide specific feedback on how students should proceed, and few existing tools are easily adaptable by instructors to suit their needs.

In the current study, we adopt an automated approach to providing feedback to students. Like many existing tools, the feedback we generate is test-case based, however we only report the total number of passing tests leaving the work of bug diagnosis and correction in the hands of the students. One goal of providing this feedback is to help students produce higher quality solutions. However, the scheduling of the feedback serves another purpose. By providing the feedback on a strictly fixed schedule to all students, rather than immediately when a student submits their work, our aim is to encourage students to start work early and manage their time effectively.

2.3 Influencing Positive Work Behaviors

Time management has been identified as one of the primary reasons that students struggle in their programming courses [36]. Moreover, procrastination is a common and widely acknowledged problem for many students [7, 18]. In computer science classrooms, there have been some deliberate efforts to reduce procrastination behavior but with limited success. In a study by Edwards et al. [12] comparing three interventions, only regular automated email alerts which included a reminder of upcoming deadlines seemed to have an effect. In another study by Ilves et al. visualizations targeting time management were only helpful for low-performing students and even harmful for performance-oriented students [22]. Novice programmers, who are typically first year students at university, are at particular risk. Students initially do not manage their time very well, and rely on explicit deadlines to control their behaviour rather than self-regulating their activity. As students progress through their degree, their time management skills tend to improve [19].

Encouraging students to start work early is well justified. Although some studies involving novice programmers have reported that the time at which a student starts their assignment has little effect on their performance [29], there is much more evidence that starting early has a positive effect [1, 15, 17, 39]. Some instructors have therefore opted for mandatory activities requiring early engagement with assignments. For example, Willman et al. [39]

included mandatory collaborative problem solving sessions and noted that providing this structure motivated students to start their assignments early.

Other work has tried to influence student time management in more subtle ways. Irwin and Edwards explored an ingenious way of using mobile gaming psychology to encourage better time management by students when working on programming assignments [23]. Specifically, they incorporated an energy bar system into their automated grading framework. Students could submit their work for feedback, but when they did so they would lose "submission energy" which took some time to regenerate. This approach was inspired by how some mobile games limit consumption by their players. The goal was to encourage students to spread their work out over time and to avoid procrastination. However, students still had to recognise for themselves that starting their work late would limit the number of submissions they could make overall. In fact, the authors found that students made significantly fewer submissions when compared with historical data, indicating that the energy bar system actually limited the number of submissions made rather than simply spreading them out over time. Similarly, the authors noted that the time at which students began work was only a few hours earlier on average compared to historical data, and a smaller difference than was hoped for.

In a similar vein, Spacco and Pugh describe an auto-grading tool called Marmoset, which allows students to submit their code which is executed against a set of public test cases [34]. If it passes these, students can choose to have their code tested using a more comprehensive set of hidden "release tests". These release tests provide very little detail about why a program failed, and students are encouraged to design their own tests to identify bugs in their code. The rate at which students can access the release tests is limited, as a token is consumed which regenerates after 24 hours. Although the authors do not explicitly state that this is designed to help students manage their time, it is an implicit incentive.

In the current study we are more explicit – we introduce a fixed schedule of deadlines which enable students to submit their work and receive automated feedback on their progress. Although submitting work by these early deadlines is completely voluntary, our goal is that the perceived value of the feedback and its fixed schedule will encourage students to engage with the assignment early.

3 METHOD

Our study was conducted in an introductory programming course for engineers at a mid-sized urban university in Auckland, New Zealand. There were 978 students in the course which was conducted over twelve teaching weeks and divided into two halves. The first half introduced students to programming using MATLAB (weeks 1–6) and the second half provided an introduction to the C programming language (weeks 7–12). No prior knowledge of programming was assumed. The assignment which is the subject of this research is from the second half of the course and was released at the beginning of week 10 (7th October) with the final submission due at the end of the course (27th October). The assignment is an individual take home assignment worth 12% of the overall grade for the course.

Table 1: Key dates and assignment events.

Event	Date & Time	Day
Assignment released	7th October	0
Early Feedback (Round 1)	19th October 1:00pm	12
Early Feedback (Round 2)	24th October 7:00am	17
Final Submission	27th October 11:59pm	20

3.1 Assignment Tasks

The assignment, a game inspired by Sokoban [38], was presented as ten smaller tasks in the context of an overarching larger project. Each task involved completing part of the functionality for an ASCII text-based console game called "The Warehouse". Students were provided with a source file (*project.c*) containing the function declarations that required implementation. A separate source file, which launched the game and made calls to the required functions, was provided to students and did not require editing. The individual tasks, along with the corresponding required functions, are listed below.

```
1. int TimeWorked(int minA, int secA, int minB, int secB)
2. int WarehouseAddress(int max)
3. void Advertise(char *words)
4. int WinningBid(int *values, int length)
5. void BoxDesign(char *design, int width, int height)
6. void WorkerRoute(int warehouse[10][10])
7-10. int MakeMove(int warehouse[10][10], char move)
```

Task 7, the `MakeMove()` function, implements the basic movement of the player in the Sokoban-style game – that is, allows a player to move a character and push boxes around a 2D grid. Tasks 8–10 extend task 7 by requiring progressively more complex cases to be handled by the `MakeMove()` function, such as adding targets and pushing multiple boxes in a single turn. The goal of the game is for the player to move all of the boxes on top of the targets.

3.2 Assignment Feedback Mechanism

Students had approximately twenty days to complete the assignment from the time it was released to the class. Prior to the final deadline, two additional scheduled deadlines were announced. Both of these "early" deadlines were optional for students, as they were ungraded, but they provided an opportunity for students to receive some early feedback on their progress. The first of these was roughly half-way between the date the assignment was released and the final deadline. Table 1 shows the key dates and times.

An online submission tool was used, and students were free to submit their code (*project.c*) at any time, and as many times as they wished. However, no feedback was provided at the time of submission. Upon making a submission, students only received a receipt comprised of a basic "confirmation" that their file had been accepted by the system. At precisely the time of each deadline, the most recent submission of a student's source file was automatically marked using a test program consisting of 200 test cases (20 per task). Students who had submitted code prior to a deadline would receive a report by email which showed a total score out of 100 and a score for each task out of 20 (the total number of tests which passed). If running the code submitted for a task resulted in either

a time-out (e.g. an infinite loop) or a crash (e.g. an invalid memory access) it was indicated in the feedback provided to the student using the code "TIMEOUT-OR-CRASH". Submissions that failed to compile were marked as "DID-NOT-COMPILE" and were awarded a total score of zero. An example of a feedback report provided to a student is given in Figure 1. For each student, the submission made closest and prior to the final submission deadline was the one used to determine their grade for the assignment.

```
YOUR PERSONAL RESULTS
——————————
– Number of tests passed for each task (out of 20 tests per task):
    – Task 1 (/20 tests) = 20
    – Task 2 (/20 tests) = 13
    – Task 3 (/20 tests) = 20
    – Task 4 (/20 tests) = 7
    – Task 5 (/20 tests) = TIMEOUT-OR-CRASH
    – Task 6 (/20 tests) = 0
    – Task 7 (/20 tests) = 0
    – Task 8 (/20 tests) = 0
    – Task 9 (/20 tests) = 0
    – Task 10 (/20 tests) = 0
================================
Final project mark (/100) = 30
================================
```

Figure 1: Feedback provided to a student for the first early submission deadline (19th October).

3.3 Data

The data we analyse is based on the log files of the assignment submission system during the C programming module of the course (Weeks 7–12), and the computed student scores after each feedback round. We identify "at-risk" students as those who received a failing score (<50%) in the MATLAB programming module (weeks 1–6), computed across laboratory, assignment and end of module test results. These students are at particular risk of failing the course overall, as the two modules contribute 50% each towards a student's final grade in the course.

4 RESULTS

We organise our results into three subsections, one for each of our research questions. In Section 4.1 we present a summary of student submission patterns and explore whether students were motivated by the early feedback deadlines. In Section 4.2 we examine the relationship between receiving early feedback and performance on the assignment, and we compare the results of students who took advantage of one or both of the early feedback rounds with those who did not. Finally, in Section 4.3 we focus on the at-risk students, and investigate how the early feedback opportunities impact their final assignment performance.

We use the term *scoring submission* (adopted from Edwards et al. [15]) to refer to a submission by a student that is processed to generate feedback for either of the two early feedback rounds or for the final assignment deadline.

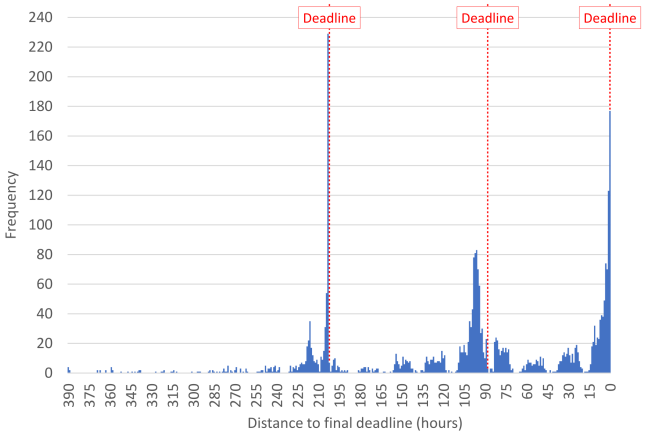

Figure 2: Frequency of first submissions by hours in advance of the final deadline.

Across all three rounds, of the 2,969 submissions made by 952 students 1,954 were scoring submissions.

4.1 Submission Patterns

Table 2 gives, for each feedback round, a summary of the number and timing of submissions relative to the corresponding deadline. Submissions made less than one hour after a deadline were counted as "late submissions" and not included in the earliest and average submission calculations presented in this table. Most of the late submissions were committed within five minutes of the deadline and, in the case of Rounds 1 and 2, all had an existing submission prior to the deadline. This suggests that the students making these late submissions may have identified a small error that was quickly rectified upon receiving feedback, or submitted the wrong file.

Figure 2 provides a histogram of the distance between the time of all scoring submissions and the final deadline grouped into equal width bins of one hour.

The red lines show the three submission deadlines (left to right: Round 1, Round 2, Final deadline). Peaks in activity can be seen close to each of the deadlines, although note the distinctly different patterns of submission for the first and second early feedback deadlines. These were set at 1:00pm and 7:00am respectively, the latter seeing much less last-minute activity.

In **Round 1**, 49% of all students made a submission. Of the 468 scoring submissions made, 26 did not compile and received a mark of zero. An additional two late submissions were made. The mean first submission time was 18 hours before the Round 1 deadline and scoring submissions on average were submitted 15 hours ahead of the deadline. On average those students who submitted in Round 1 made 1.40 submissions, with 122 students making more than one submission. The longest time between a first and scoring submission recorded for an individual student in Round 1 was five days and three and a half hours. The maximum number of submissions made by an individual was eight.

In **Round 2**, 70% of all students made a submission. Of the 670 scoring submissions, 25 failed to compile. In total 1039 submissions

Event	Students	Submissions	Earliest	Average First Submission	Average Scoring Submission
Round 1	468	655	7 days 19 hours early	0 days 18 hours early	15 hours 0 minutes early
Round 2	670	1039	5 days 5 hours early	1 day 5 hours early	20 hours 40 minutes early
Final	816	1275	3 days 14 hours early	1 day 4 hours early	21 hours 41 minutes early

Table 2: Submissions relative to deadlines.

were made with 1.6 submissions per student on average. Five students had scoring submissions that failed to compile in both Rounds 1 and 2. Of the students who submitted in Round 1, 51 did not make a submission to Round 2. In this round, 38% of scoring submissions were from students who had not submitted in Round 1.

For the **Final deadline**, 1275 submissions were made by 816 students. Of these, 231 students (28%) had not submitted to the early feedback rounds. Only 13 students (1.6%) of the students who submitted to the final round made a scoring submission that failed to compile. These non-compiling solutions were all first time submissions.

A total of 952 students received a mark for the assignment – that is, they made at least one submission at some point before the final deadline.

4.2 Early Feedback and Performance

There is a clear relationship between receiving early feedback and final performance on the assignment. The average final assignment mark for students who made their first submission before the first early feedback deadline was **95.8%** (n = 468, SD=9.3), compared to **89.8%** (n = 253, SD=14.7) for students who initially submitted prior to the second early feedback deadline. Students who only submitted after the Round 2 deadline, and therefore did not receive any early feedback, earned an average mark for the assignment of **64.3%** (n = 231, SD=29.3). A Kruskal-Wallis rank sum test indicates there is a statistically significant difference in assignment scores between these three groups (χ^2 = 343.68, p < .001).

Twenty four students made a scoring submission to Round 1 of 100%, and of these, only two resubmitted in Round 2 and one in the final round. Figure 3 shows the time to deadline for scoring submissions, for all three rounds, with submissions grouped by grade. A clear trend is observable across each round, where submissions that received a lower grade were more likely to have been made closer to the deadline. Interestingly, as a result of the early morning deadline (7:00am) for the second round of feedback, few submissions were made immediately leading up to the deadline (hence the "Round 2" boxes are elevated from the x-axis).

4.2.1 Performance by task. Figure 4 shows the average mark by task (out of the 10 tasks for the assignment) and round for the 318 students who submitted a scoring submission prior to both of the early rounds and prior to the final deadline. These students, who received the automated feedback for both of the early feedback rounds, were able to progressively improve their performance on every task in the assignment across their three submissions. A contributing factor to this trend is that students were more likely to submit incomplete work for the earlier rounds.

Not unsurprisingly, the more difficult the task the more valuable the early submission feedback appears to be. The students found

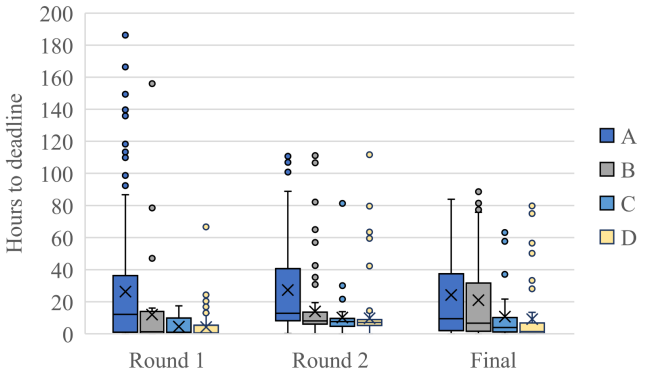

Figure 3: Scoring submissions grouped based on grade level by time of submission, in hours, prior to event deadline. Grades: A (90-100%); B (65-79%); C (50-64%); D (0-49%).

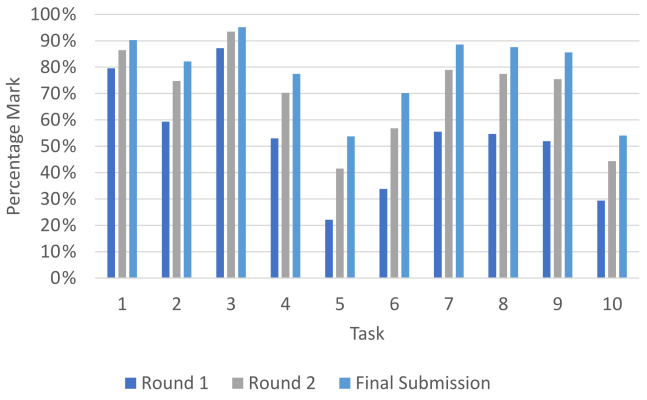

Figure 4: Average mark by task and round for students who submitted to all three rounds.

Task 10, as measured by performance, the hardest task. In Round 1 only 57 students (18%) gave a fully correct answer for Task 10 but by Round 2 this figure had increased to 142, and in the final submission 54% of students gave a fully correct answer for Task 10.

4.2.2 Pathways to completion. Figure 5 illustrates the pathways through which students moved across the three submission deadlines. The students are grouped into three rows (or bands) in the y-axis based on the mark their submission achieved, while the x-axis (or columns) represents the three consecutive deadlines. The circle size is proportional to the number of students making a submission within that grade band. The lines indicate movement of students between the various grade bands and rounds, moving left to right. The line thickness indicates the proportion of students

Table 3: Assignment performance of "at-risk" students (who failed prior module), classified by first submission round.

Event	*n*	Prior module avg. (SD)	Assignment avg. (SD)
Round 1	6	39.2% (9.2)	72.7% (27.6)
Round 2	15	40.3% (7.0)	69.5% (26.1)
Final	61	37.9% (8.7)	40.2% (30.1)

making a particular shift. While most students who scored in the top third continued through to the final submission within that grade band, a small number were happy with their grade and chose not to submit at Round 2. All students were required to submit by the final deadline in order to receive a mark for the assignment. Of those students whose first submission scored <=33%, 77% were able to improve on their first submission in Round 2.

4.3 Effects for At-Risk Students

A total of 82 students failed the first course module (Weeks 1–6) and were therefore classified as "at-risk". This group of students achieved an average mark of 38.4% on the first module. Table 3 breaks down this group of students based on the feedback round to which they made their first submission, and shows the average mark ultimately achieved on the assignment. Amongst this at-risk group, 25% submitted to at least one of the early feedback rounds. There is no difference in prior module performance for these students based on the round to which they first submitted (Kruskal-Wallis χ^2 = 0.794, p = .672). In contrast, when comparing performance on the final assignment, students who received at least some feedback from an early round performed significantly better (Kruskal-Wallis χ^2 = 16.392, p < .001).

5 DISCUSSION

With regards to RQ1, there is evidence that the voluntary feedback opportunities were motivating for many students. Around half of the students in the course made a submission more than a week prior to the final assignment deadline, and the submission patterns in Figure 2 show a distinctive ramping up in activity in the days

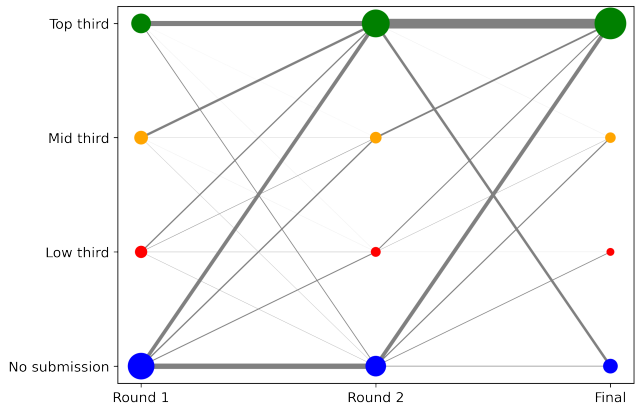

Figure 5: Pathways to completion: top third (mark > 66); mid third (33 < mark <= 66), low third (mark <= 33)

prior to each deadline. This suggests that many students perceived value in the feedback, and submitted their code earlier than they otherwise would have. Indeed, several responses by students on the end of course evaluations indicated the early feedback prompted them to engage with the assignment, for example:

> *"I cannot express how helpful for my learning the early submission feedback was for me. Not only does it give me feedback when I am all alone without giving any concrete hints, it helped me stop procrastinating till the last minute".*

In answer to RQ2, the relationship between receiving early feedback and subsequent assignment performance was clear – students who submitted prior to the early feedback deadlines performed significantly better than students who did not, and the average score for each assignment task increased from one round to the next. It is also interesting to note that the 13 students who received zero for the assignment (as a result of submitting non-compiling code) did not receive any early feedback.

Finally, with respect to RQ3, at-risk students benefited greatly if they submitted their work early. The 25% who chose to take advantage of at least one early feedback round were not necessarily the top performing of the at-risk students, scoring no better overall on the first course module (see Table 3). However, they scored significantly better on the assignment by the time of the final deadline. This may be a consequence of being prompted to start the assignment earlier, thus getting an initial sense of how much time the assignment would take them, and how they should allocate their available time across the tasks.

5.1 Limitations

We make the claim that the early feedback deadlines encouraged students to start work earlier on their programming assignments. We see an impact on submission behaviour, with an obvious sharp increase in activity immediately prior to each deadline. However, we haven't directly collected data on when students actually started work, only on when they submitted their code for feedback.

We also have no data on how students made use of the feedback, however we suspect that once students passed all of the tests for a task they would no longer allocate time to working on that task. In some cases, given the nature of the feedback, it may have been frustrating for students who failed just one or two tests as they were not given an indication of which tests failed. We did not investigate how understandable this simpler feedback is or what challenges this form of feedback might pose for the students, this is a potential area for future work.

Finally, there was no feedback generated on code quality, nor did we analyse any stylistic elements of the submitted code. This may have resulted in the submission of functionally correct, but poorly organised code, and future work incorporating some automatic feedback on code style is warranted [6]. Future work should also explore the effects of variations in the feedback provided, such as including details of one or more failing test cases.

6 CONCLUSIONS

In this work we explored the use of optional early feedback deadlines, allowing students to receive automated feedback on the correctness of their code, as a way to promote early engagement with

a programming assignment. We found that these voluntary feedback opportunities were highly motivating for some students, with half of the class submitting their work more than a week before the final deadline for grading. In addition, the feedback appears to have been useful in helping students improve their scores despite its simple format. Basic feedback about the degree of correctness, broken down by task to allow for some high-level localisation of issues, is sufficient for helping most novice programmers improve their solutions. Finally, like much prior literature on novice programmers, we find that students who start an assignment early, as indicated by early scoring submissions, are more likely to receive a high grade on the assignment.

REFERENCES

[1] Tapio Auvinen. 2015. Harmful Study Habits in Online Learning Environments with Automatic Assessment. In *2015 International Conference on Learning and Teaching in Computing and Engineering*. 50–57.

[2] Brett A. Becker, Paul Denny, Raymond Pettit, Durell Bouchard, Dennis J. Bouvier, Brian Harrington, Amir Kamil, Amey Karkare, Chris McDonald, Peter-Michael Osera, Janice L. Pearce, and James Prather. 2019. Compiler Error Messages Considered Unhelpful: The Landscape of Text-Based Programming Error Message Research. In *Proceedings of the Working Group Reports on Innovation and Technology in Computer Science Education (ITiCSE-WGR '19)*. ACM, New York, NY, USA, 177–210. https://doi.org/10.1145/3344429.3372508

[3] Brent J. Bowman and William A. Newman. 1990. Software Metrics as a Programming Training Tool. *J. Syst. Softw.* 13, 2 (Oct. 1990), 139–147. https://doi.org/10.1016/0164-1212(90)90119-7

[4] Rachel Cardell-Oliver. 2011. How Can Software Metrics Help Novice Programmers?. In *Proceedings of the Thirteenth Australasian Computing Education Conference - Volume 114 (ACE '11)*. Australian Computer Society, Inc., AUS, 55–62.

[5] Rachel Cardell-Oliver, Lu Zhang, Rieky Barady, You Hai Lim, Asad Naveed, and Terry Woodings. 2010. Automated Feedback for Quality Assurance in Software Engineering Education. In *Proceedings of the 2010 21st Australian Software Engineering Conference (ASWEC '10)*. IEEE Computer Society, USA, 157–164. https://doi.org/10.1109/ASWEC.2010.24

[6] Rohan Roy Choudhury, HeZheng Yin, Joseph Moghadam, and Armando Fox. 2016. AutoStyle: Toward Coding Style Feedback At Scale. In *Proceedings of the 19th ACM Conference on Computer Supported Cooperative Work and Social Computing Companion (CSCW '16 Companion)*. Association for Computing Machinery, New York, NY, USA, 21–24. https://doi.org/10.1145/2818052.2874315

[7] Sophie H. Cormack, Laurence A. Eagle, and Mark S. Davies. 2020. A large-scale test of the relationship between procrastination and performance using learning analytics. *Assessment & Evaluation in Higher Education* 0, 0 (2020), 1–14. https://doi.org/10.1080/02602938.2019.1705244 arXiv:https://doi.org/10.1080/02602938.2019.1705244

[8] Paul Denny, Brett A. Becker, Michelle Craig, Greg Wilson, and Piotr Banaszkiewicz. 2019. Research This! Questions That Computing Educators Most Want Computing Education Researchers to Answer. In *Proceedings of the 2019 ACM Conference on International Computing Education Research (ICER '19)*. Association for Computing Machinery, New York, NY, USA, 259–267. https://doi.org/10.1145/3291279.3339402

[9] Paul Denny, Andrew Luxton-Reilly, Michelle Craig, and Andrew Petersen. 2018. Improving Complex Task Performance Using a Sequence of Simple Practice Tasks. In *Proceedings of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE 2018)*. Association for Computing Machinery, New York, NY, USA, 4–9. https://doi.org/10.1145/3197091.3197141

[10] Paul Denny, Andrew Luxton-Reilly, Ewan Tempero, and Jacob Hendrickx. 2011. CodeWrite: Supporting Student-Driven Practice of Java. In *Proceedings of the 42nd ACM Technical Symposium on Computer Science Education (SIGCSE '11)*. Association for Computing Machinery, New York, NY, USA, 471–476. https://doi.org/10.1145/1953163.1953299

[11] Paul Denny, James Prather, and Brett A. Becker. 2020. Error Message Readability and Novice Debugging Performance. In *Proceedings of the 2020 ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE '20)*. Association for Computing Machinery, New York, NY, USA, 480–486. https://doi.org/10.1145/3341525.3387384

[12] Stephen H. Edwards, Joshua Martin, and Clfford A. Shaffer. 2015. Examining Classroom Interventions to Reduce Procrastination. In *Proceedings of the 2015 ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE '15)*. Association for Computing Machinery, New York, NY, USA, 254–259. https://doi.org/10.1145/2729094.2742632

[13] Stephen H. Edwards and Krishnan Panamalai Murali. 2017. CodeWorkout: Short Programming Exercises with Built-in Data Collection. In *Proceedings of the 2017 ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE '17)*. Association for Computing Machinery, New York, NY, USA, 188–193. https://doi.org/10.1145/3059009.3059055

[14] Stephen H. Edwards and Manuel A. Perez-Quinones. 2008. Web-CAT: Automatically Grading Programming Assignments. In *Proceedings of the 13th Annual Conference on Innovation and Technology in Computer Science Education (ITiCSE '08)*. Association for Computing Machinery, New York, NY, USA, 328. https://doi.org/10.1145/1384271.1384371

[15] Stephen H. Edwards, Jason Snyder, Manuel A. Pérez-Quiñones, Anthony Allevato, Dongkwan Kim, and Betsy Tretola. 2009. Comparing Effective and Ineffective Behaviors of Student Programmers. In *Proceedings of the Fifth International Workshop on Computing Education Research Workshop (ICER '09)*. Association for Computing Machinery, New York, NY, USA, 3–14. https://doi.org/10.1145/1584322.1584325

[16] Tommy Färnqvist and Fredrik Heintz. 2016. Competition and Feedback through Automated Assessment in a Data Structures and Algorithms Course. In *Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE '16)*. Association for Computing Machinery, New York, NY, USA, 130–135. https://doi.org/10.1145/2899415.2899454

[17] James B. Fenwick, Cindy Norris, Frank E. Barry, Josh Rountree, Cole J. Spicer, and Scott D. Cheek. 2009. Another Look at the Behaviors of Novice Programmers. In *Proceedings of the 40th ACM Technical Symposium on Computer Science Education (SIGCSE '09)*. Association for Computing Machinery, New York, NY, USA, 296–300. https://doi.org/10.1145/1508865.1508973

[18] Yoshiko Goda, Masanori Yamada, Hiroshi Kato, Takeshi Matsuda, Yutaka Saito, and Hiroyuki Miyagawa. 2015. Procrastination and other learning behavioral types in e-learning and their relationship with learning outcomes. *Learning and Individual Differences* 37 (2015), 72 – 80. https://doi.org/10.1016/j.lindif.2014.11.001

[19] Keith Gregory and Sue Morón-García. 2009. Assignment submission, student behaviour and experience. *Engineering Education* 4, 1 (2009), 16–28. https://doi.org/10.11120/ened.2009.04010016

[20] Petri Ihantola, Tuukka Ahoniemi, Ville Karavirta, and Otto Seppälä. 2010. Review of Recent Systems for Automatic Assessment of Programming Assignments. In *Proceedings of the 10th Koli Calling International Conference on Computing Education Research (Koli Calling '10)*. Association for Computing Machinery, New York, NY, USA, 86–93. https://doi.org/10.1145/1930464.1930480

[21] Petri Ihantola, Arto Vihavainen, Alireza Ahadi, Matthew Butler, Jürgen Börstler, Stephen H. Edwards, Essi Isohanni, Ari Korhonen, Andrew Petersen, Kelly Rivers, Miguel Ángel Rubio, Judy Sheard, Bronius Skupas, Jaime Spacco, Claudia Szabo, and Daniel Toll. 2015. Educational Data Mining and Learning Analytics in Programming: Literature Review and Case Studies. In *Proceedings of the 2015 ITiCSE on Working Group Reports (ITICSE-WGR '15)*. Association for Computing Machinery, New York, NY, USA, 41–63. https://doi.org/10.1145/2858796.2858798

[22] Kalle Ilves, Juho Leinonen, and Arto Hellas. 2018. Supporting self-regulated learning with visualizations in online learning environments. In *Proceedings of the 49th ACM Technical Symposium on Computer Science Education*. 257–262.

[23] Michael S. Irwin and Stephen H. Edwards. 2019. Can Mobile Gaming Psychology Be Used to Improve Time Management on Programming Assignments?. In *Proceedings of the ACM Conference on Global Computing Education (CompEd '19)*. Association for Computing Machinery, New York, NY, USA, 208–214. https://doi.org/10.1145/3300115.3309517

[24] Lucy Jiang, Robert Rewcastle, Paul Denny, and Ewan Tempero. 2020. CompareCFG: Providing Visual Feedback on Code Quality Using Control Flow Graphs. In *Proceedings of the 2020 ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE '20)*. Association for Computing Machinery, New York, NY, USA, 493–499. https://doi.org/10.1145/3341525.3387362

[25] Ayaan M. Kazerouni, Stephen H. Edwards, and Clifford A. Shaffer. 2017. Quantifying Incremental Development Practices and Their Relationship to Procrastination. In *Proceedings of the 2017 ACM Conference on International Computing Education Research (ICER '17)*. Association for Computing Machinery, New York, NY, USA, 191–199. https://doi.org/10.1145/3105726.3106180

[26] Hieke Keuning, Johan Jeuring, and Bastiaan Heeren. 2018. A Systematic Literature Review of Automated Feedback Generation for Programming Exercises. *ACM Trans. Comput. Educ.* 19, 1, Article Article 3 (Sept. 2018), 43 pages. https://doi.org/10.1145/3231711

[27] Pardha Koyya, Lee Young, and Jeong Yang. 2013. Feedback for Programming Assignments Using Software-Metrics and Reference Code. *ISRN Software Engineering* (2013). https://doi.org/10.1155/2013/805963

[28] Lauri Malmi, Ville Karavirta, Ari Korhonen, and Jussi Nikander. 2005. Experiences on Automatically Assessed Algorithm Simulation Exercises with Different Resubmission Policies. *J. Educ. Resour. Comput.* 5, 3 (Sept. 2005), 7–es. https://doi.org/10.1145/1163405.1163412

[29] Keir Mierle, Kevin Laven, Sam Roweis, and Greg Wilson. 2005. Mining Student CVS Repositories for Performance Indicators. In *Proceedings of the 2005 International Workshop on Mining Software Repositories (MSR '05)*. Association for Computing Machinery, New York, NY, USA, 1–5. https://doi.org/10.1145/1083142.1083150

[30] Nick Parlante. 2007. Nifty Reflections. *SIGCSE Bull.* 39, 2 (June 2007), 25–26. https://doi.org/10.1145/1272848.1272876

[31] James Prather, Brett A. Becker, Michelle Craig, Paul Denny, Dastyni Loksa, and Lauren Margulieux. 2020. What Do We Think We Think We Are Doing?: Metacognition and Self-Regulation in Programming. In *Proceedings of the 2020 ACM Conference on International Computing Education Research (ICER '20)*. Association for Computing Machinery, New York, NY, USA, 12. https://doi.org/10.1145/3372782.3406263

[32] Paul Ramsden. 1992. *Learning to Teach in Higher Education.* Routledge, London.

[33] Jaime Spacco, Paul Denny, Brad Richards, David Babcock, David Hovemeyer, James Moscola, and Robert Duvall. 2015. Analyzing Student Work Patterns Using Programming Exercise Data. In *Proceedings of the 46th ACM Technical Symposium on Computer Science Education (SIGCSE '15)*. Association for Computing Machinery, New York, NY, USA, 18–23. https://doi.org/10.1145/2676723.2677297

[34] Jaime Spacco and William Pugh. 2006. Helping Students Appreciate Test-Driven Development (TDD). In *Companion to the 21st ACM SIGPLAN Symposium on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA '06)*. Association for Computing Machinery, New York, NY, USA, 907–913. https://doi.org/10.1145/1176617.1176743

[35] Claudia Szabo, Nickolas Falkner, Antti Knutas, and Mohsen Dorodchi. 2018. Understanding the Effects of Lecturer Intervention on Computer Science Student Behaviour. In *Proceedings of the 2017 ITiCSE Conference on Working Group Reports (ITiCSE-WGR '17)*. Association for Computing Machinery, New York, NY, USA, 105–124. https://doi.org/10.1145/3174781.3174787

[36] Rebecca Vivian, Katrina Falkner, and Nickolas Falkner. 2013. Computer Science Students' Causal Attributions for Successful and Unsuccessful Outcomes in Programming Assignments. In *Proceedings of the 13th Koli Calling International Conference on Computing Education Research (Koli Calling '13)*. Association for Computing Machinery, New York, NY, USA, 125–134. https://doi.org/10.1145/2526968.2526982

[37] Jacqueline L. Whalley and Anne Philpott. 2011. A Unit Testing Approach to Building Novice Programmers' Skills and Confidence. In *Proceedings of the Thirteenth Australasian Computing Education Conference - Volume 114 (ACE '11)*. Australian Computer Society, Inc., AUS, 113–118. https://dl.acm.org/doi/10.5555/2459936.2459950

[38] Wikipedia. 2020. Sokoban – Wikipedia. https://en.wikipedia.org/wiki/Sokoban

[39] Salla Willman, Rolf Lindén, Erkki Kaila, Teemu Rajala, Mikko-Jussi Laakso, and Tapio Salakoski. 2015. On study habits on an introductory course on programming. *Computer Science Education* 25, 3 (2015), 276–291. https://doi.org/10.1080/08993408.2015.1073829