

## Towards understanding the effective design of automated formative feedback for programming assignments

Qiang Hao, David H. Smith IV, Lu Ding, Amy Ko, Camille Ottaway, Jack Wilson, Kai H. Arakawa, Alistair Turcan, Timothy Poehlman & Tyler Greer

To cite this article: Qiang Hao, David H. Smith IV, Lu Ding, Amy Ko, Camille Ottaway, Jack Wilson, Kai H. Arakawa, Alistair Turcan, Timothy Poehlman & Tyler Greer (2021): Towards understanding the effective design of automated formative feedback for programming assignments, *Computer Science Education*, DOI: [10.1080/08993408.2020.1860408](https://doi.org/10.1080/08993408.2020.1860408)

To link to this article: <https://doi.org/10.1080/08993408.2020.1860408>



Published online: 31 Jan 2021.



Submit your article to this journal [↗](#)



Article views: 165



View related articles [↗](#)




View Crossmark data [↗](#)



Citing articles: 1 View citing articles [↗](#)



## Towards understanding the effective design of automated formative feedback for programming assignments

Qiang Hao <sup>a</sup>, David H. Smith IV<sup>a</sup>, Lu Ding<sup>b</sup>, Amy Ko<sup>c</sup>, Camille Ottaway<sup>a</sup>, Jack Wilson<sup>a</sup>, Kai H. Arakawa<sup>a</sup>, Alistair Turcan<sup>a</sup>, Timothy Poehlman<sup>a</sup> and Tyler Greer<sup>a</sup>

<sup>a</sup>Computer Science & SMATE, Western Washington University, Bellingham, WA, US; <sup>b</sup>Faculty Development and Innovation Center, Eastern Illinois University, Charleston, IL, US; <sup>c</sup>The Information School, University of Washington, Seattle, WA, US

### ABSTRACT

**Background and Context:** automated feedback for programming assignments has great potential in promoting just-in-time learning, but there has been little work investigating the design of feedback in this context.

**Objective:** to investigate the impacts of different designs of automated feedback on student learning at a fine-grained level, and how students interacted with and perceived the feedback.

**Method:** a controlled quasi-experiment of 76 CS students, where students of each group received a different combination of three types of automated feedback for their programming assignments.

**Findings:** feedback addressing the gap between expected and actual outputs is critical to effective learning; feedback lacking enough details may lead to system gaming behaviors.

**Implications:** the design of feedback has substantial impacts on the efficacy of automated feedback for programming assignments; more research is needed to extend what is known about effective feedback design in this context.

### ARTICLE HISTORY

Received 1 April 2020

Accepted 3 December 2020

### KEYWORDS

Automated feedback;  
formative feedback;  
programming assignments;  
system gaming behavior

## 1. Introduction

Providing timely feedback to student programming assignments is a challenging task for computing educators. On the one hand, the nature of code makes programming assignments challenging to assess. For the same programming assignment, many problem-solving strategies may be available for students to use, and very different code may achieve the same purpose (Kinnunen & Simon, 2012). Different individual coding habits and styles only exacerbate the challenge. Instructors typically need to test a student's code against a large number of testing cases to draw a solid conclusion on its correctness. On the other hand, the fast increasing CS enrollment substantially lowers instructor-to-student ratios and increases the workload of instructors (Camp et al., 2017; Greer et al., 2019; Sax et al., 2017). For instance, undergraduate CS enrollment has doubled from 2011 to 2017 in U.S. colleges, while the number of CS instructors grows at a significantly slower rate (Computing Research Association, 2017; Hao et al., 2019a). As a result, it is more

**CONTACT** Qiang Hao Email  [qiang.hao@wwu.edu](mailto:qiang.hao@wwu.edu)  Western Washington University, Bellingham, WA, US.  
Present affiliation for Camille Ottaway is FDSST Public Schools, Denver, Colorado.

challenging for CS instructors to allocate sufficient time to assess and provide feedback for programming assignments.

Feedback can be a powerful learning intervention when used effectively. Decades of educational research have demonstrated consistent efficacy of feedback in improving student learning performance. Depending on the delivery time, feedback can be classified into two types: summative and formative feedback. Summative feedback is provided to learners along with the assessment results, while formative feedback is provided during the learning process before the assessment results are given (Brinko, 1993; Gielen et al., 2010). Formative feedback has been constantly found to be more effective than summative feedback across different fields (Saifi et al., 2011; Yorke, 2001, 2003). (Black & Wiliam, 2009) reported that formative feedback, as one of the most effective educational interventions, produced learning gains with stable effect sizes between 0.4 and 0.7. Formative feedback works better because it can help learners understand where they are in their learning, where they are going, and how to get there (Nicol & Macfarlane-Dick, 2006; Sadler, 1989).

Given the benefits of formative feedback, computing educators and researchers have studied how to automate the process of grading programming assignments and providing formative feedback to students for more than two decades. Automated grading systems have been developed, implemented, and evaluated (Chen, 2004; Keuning et al., 2018; Parihar et al., 2017). These systems benefit both students and instructors. Using these systems, students can submit their programming assignments numerous times before the deadlines and get feedback for further improvements to their code or fixing mistakes, promoting just-in-time learning (Chow et al., 2017; Ihanola et al., 2010; Parihar et al., 2017; Pears et al., 2007). These systems can also allow instructors to save substantial time from grading programming assignments manually, and focus on the pedagogical design of the courses (Cheang et al., 2003; Pears et al., 2007; Vujošević -Janičić et al., 2013).

Despite the contributions from prior studies on automated feedback, few studies have explored the design of formative feedback for programming assignments empirically. Automated feedback is powerful, but also substantially less adaptive compared with human helpers. When human helpers are present, back-and-forth communication is used to help novice learners by clarifying confusions, adapting student questions, and diagnosing student learning gaps (Hao et al., 2016; Smith IV et al., 2020). In contrast, automated feedback is typically delivered as information for learners to digest and absorb with little to no communication capacity. As a result, the information contained in automated feedback needs to be carefully designed to realize its full potential. In addition, proposals to limit information in automated feedback need further verification from empirical studies. For instance, a widespread concern with automated feedback systems is that formative feedback may encourage students to abuse the automated systems (Chen, 2004; Guerreiro & Georgouli, 2006; Ihanola et al., 2010). One common practice to address this issue is to limit the information contained in automated formative feedback. However, there is little evidence supporting the efficacy of such practices. Understanding the designs of feedback based on solid evidence is critical to strengthening the effectiveness of automated feedback systems.

To fill this gap, this study explored the efficacy of different types of formative feedback on student learning in the context of automated feedback for programming assignments through a controlled quasi-experiment. This study contributes to a deeper understanding

of the designs of automated feedback, which in turn informs the design of automated feedback systems.

## 2. Background

### 2.1. Design of formative feedback

Feedback, as a powerful educational intervention, has been studied extensively in the last three decades (Hattie & Gan, 2011; Van der Kleij et al., 2015). Numerous models of feedback were proposed from different perspectives, such as the role of feedback in the learning process, the impacts of feedback on student performance, classification of feedback, and contributing factors to effective feedback (Boud & Molloy, 2013; Butler & Winne, 1995; Evans, 2013; Ihantola et al., 2015; Kluger & DeNisi, 1996; Pardo, 2018; Shute, 2008).

One of the most comprehensive conceptual analysis of feedback was provided by Hattie and Timperley (2007). The authors proposed a feedback model centered around reducing the gap between students' actual performance and expected performance through three questions: where am I going, how am I going, and where to next. Based on the model, the authors further proposed a feedback classification with four levels: task, process, regulation, and self. Feedback at the task level is about how well a task is being accomplished or performed. An example of feedback at this level is the information on whether an answer is correct. Feedback at the process level defines what is needed in order to finish the task. Hints can generally be considered feedback at the process level. Feedback at the regulation level refers to suggestions on how to self-regulate the learning process. Feedback at the self level is about the characteristics of the learner. Feedback at the self level has been found to be ineffective for student learning by many studies, because it fails to address how to achieve the intended learning goals (Hattie & Timperley, 2007; Van der Kleij et al., 2012). It is worth noting that both the model and feedback classification proposed by Hattie and Timperley (2007) are tailored to learning and teaching in classroom settings, emphasizing the communication between students and teachers. When the learning context changes and requires the interaction between humans and machines, the proposed feedback system may not be perfectly applicable. For instance, it is still tremendously difficult for any intelligent learning systems to perform back-and-forth communications with learners, let alone providing useful feedback at the regulation and self levels.

The development of learning technologies enabled feedback to be provided automatically in a scalable approach. However, automated feedback tends to lack the ability to adapt to students on an individual basis in the way that human helpers can. Therefore, how to optimize the design of automated feedback in the context of computer-based instruction attracted substantial attention from researchers. Narciss and Huth (2004) proposed a notable feedback classification specific to learning tasks in this context. Three levels of feedback are included in this classification, including knowledge of results (KR), knowledge of correct responses (KCR), and elaborated feedback (EF). KR feedback refers to the information on correctness. For example, error flagging that only shows where the error is located but does not show additional information is a type of KR. KCR feedback provides the correct answer to the problem. EF feedback provides a detailed explanation and even hints to fix the identified problems in student answers. Hundreds of

studies adopted this feedback classification to investigate the impacts of varied forms of feedback on student learning (Lee et al., 2010; McMillan et al., 2013). Van der Kleij et al. (2015) synthesized 40 studies that adopted this feedback classification, and found that EF in particular was more effective than KR and KCR for higher-order learning outcomes, and confirmed that the feedback classification proposed by Narciss and Huth (2004) provided a theoretical foundation for studying the designs of feedback in the context of computer-based instruction. This perspective of seeing feedback as information that can be designed has profound impacts on research on feedback, learning systems, and intelligent tutors across different educational fields (e.g., Alvarez et al., 2012; Fyfe & Rittle-Johnson, 2016; Van der Kleij et al., 2015).

Additionally, a large body of research has investigated what characteristics of feedback contribute to its effectiveness from the perspective of information design. Three factors, including student prior knowledge, tasks, and the design of feedback were found most important for the efficacy of feedback (Narciss & Huth, 2004; Narciss et al., 2014; Shute, 2008). In order for feedback to be effective, four conditions need to be met: 1) the students need the feedback, 2) the students have sufficient time to process the feedback, 3) the students are willing to use the feedback, and 4) the students are able to make sense of the feedback (Bangert-Drowns et al., 1991; Stobart, 2008). Different designs of feedback may also impact its efficacy. For instance, specificity is one of the design aspects that have been studied intensively. Specificity is needed for effective information delivery. Feedback lacking specificity is more likely to result in frustration and loss of motivation (Van Merriënboer & Sweller, 2005). Nevertheless, feedback that is too specific and complex may also risk overwhelming students (Serge et al., 2013).

## **2.2. Auto-grading and auto-feedback systems**

Programming assignments are challenging to grade and to provide feedback on manually. When class sizes grow fast, this problem can be even more challenging to tackle. Therefore, how to automate the process of grading assignments and providing feedback to students has attracted many researchers from different fields, such as computing education and software engineering Cassel and Fox (2000).

Early studies on the topic of automated grading and feedback mainly focused on developing and testing automated grading systems because performing assessments in a timely manner is more important for instructors to perform their essential duties than to provide informative feedback. Therefore, the focus on automated feedback was sidelined at first. Furthermore, many studies expressed concerns that automation may incite students to abuse the feedback system (Cheang et al., 2003; Chen, 2004; Guerreiro & Georgouli, 2006; Ihanola et al., 2010). Several measurements were proposed to discourage potential “system gaming behaviors”, such as providing only summative feedback, limiting the number of submission attempts, and limiting the feedback content (Chow et al., 2017; Daly & Horgan, 2004; Keuning et al., 2018; Pieterse, 2013). System gaming behaviors are a reasonable concern if hints revealing direct answers can be easily accessed by students. However, feedback is not only limited to hints. For instance, correctness, as a type of feedback, has been found positively impactful to learning if it is given before the summative assessment (Ala-Mutka, 2005; Alemán, 2010).

Continued research on this topic in the last decade led to the development of a number of tools focusing on providing students with feedback on their programming process. How to effectively generate automated feedback was the major focus of these studies (Luxton-Reilly et al., 2018). Web-CAT, a representative system developed in the early days, has evolved into an online automated feedback system that served more than 30 institutions in the U.S. (Edwards & Perez-Quinones, 2008). Bluefix, an extension of BlueJ IDE, provided programming students error diagnosis and repair feedback based on crowd-sourced information (Watson et al., 2012). Neve et al. (2012) attempted to provide in-lab tutor-student dynamic interaction specific to programming through an online coding environment. Additionally, research on enhanced compiler error messages and student misconceptions also provided valuable findings that contribute to this topic (Becker et al., 2019). For instance, Karvelas et al. (2020) found that compilation mechanisms and error message presentation had substantial impacts on the programming behaviors of novice learners, which indicates that more human-computer interaction research on the design of automated feedback systems is needed. Hao and Tsikerdekis (2019) studied using the existing infrastructure of continuous integration tools to provide students with automated feedback and attempted to examine its impacts on student knowledge transfer. Gusukuma et al. (2020) studied decoupling the modules of conventional automated feedback systems such as type inferencing, flow analysis, pattern matching, and unit testing.

Recent studies on the topic of automated feedback have shifted the focus to studying data-driven approaches to test and provide feedback on students' code (Chow et al., 2017; Parihar et al., 2017; Price et al., 2018; Vujošević -Janičić et al., 2013). The data-driven approaches include clustering, filtering, and pattern mining (Gao et al., 2016; Head et al., 2017). Such approaches typically depend on a massive number of student submissions, and aim to provide students suggestions on repairing their code by measuring the distance between exemplary code and individual students' code (Drummond et al., 2014; Gulwani et al., 2018; Wang et al., 2018). Such approaches still require testing in an authentic environment and need further investigation on their reliability on smaller datasets. CLuster And RepAir tool (CLARA) is a representative system investigated by recent studies on this topic. CLARA leverages the "wisdom of the crowd" by extracting programs that pass test cases, clustering them based on approach similarity, comparing working programs to nonworking ones (Gulwani et al., 2018). Although CLARA was tested in the context of MOOCs where a massive number of student code submissions are available, whether it works in the context of face-to-face classrooms is still unknown.

Despite the advances in feedback generation approaches, few studies investigated the designs of feedback in the context of computing education from the perspective of information design. There is strong evidence from other disciplines that feedback can be designed and delivered in different ways, and such differences may render different influences on student performance (Alvarez et al., 2012; Fyfe & Rittle-Johnson, 2016; Van der Kleij et al., 2015). The meta-analysis study of Keuning et al. (2018) was the first that investigated automated feedback using the perspective of information design in computing education, where Keuning et al. (2018) attempted to classify existing automated feedback tools using the feedback classification proposed by Narciss and Huth (2004). Studies investigating a specific design of automated feedback design are abundant (Hundhausen & Brown, 2007; Nygren et al., 2019). However, empirical studies that

compared different automated feedback designs from the perspective of information design are rare in computing education (Luxton-Reilly et al., 2018). Hao et al. (2019b) examined the efficacy of different designs of automated feedback for programming assignments by comparing the efficacy of correctness, knowledge gap, and hints on student comprehension and problem-solving of complex programming assignments, and found addressing the knowledge gap is critical to improve student learning. However, their findings were limited by the small sample size and a lack of theoretical framework. A deeper understanding of how different types of feedback impact student learning is critical to guide system design and development, and is likely to have direct impacts on student learning (Keuning et al., 2018).

### 3. Research design

#### 3.1. Research question

The following questions guided our study:

- (1) How do different types of automated formative feedback impact student performance on programming assignments?
- (2) How do different types of automated formative feedback impact student interaction with the automated feedback system?
- (3) How do students perceive different types of automated formative feedback?

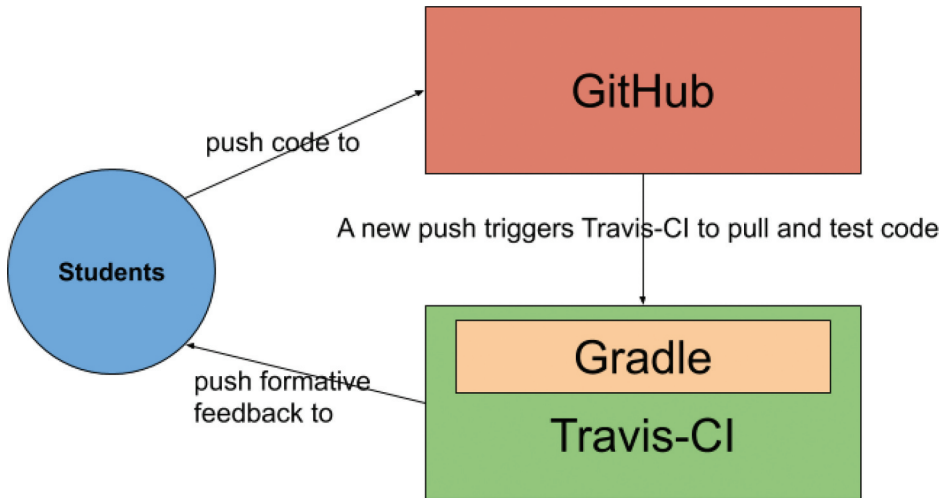
#### 3.2. System design

For this study, we implemented an automated formative feedback system utilizing three technologies that are popular with programming courses, including GitHub, Gradle, and Travis-CI. GitHub (github.com) is a version-control code repository hosting service. We used GitHub to host student programming assignments. Students were instructed to commit and push their code once they reached a self-determined milestone from the beginning of the course. Gradle (gradle.org) is an open-source build automation system for Java programming. Travis-CI (travis-ci.com) is a continuous integration service used to build and test software projects hosted at GitHub. We used Gradle and Travis-CI to build, test, and provide feedback to student programming assignments hosted on GitHub.

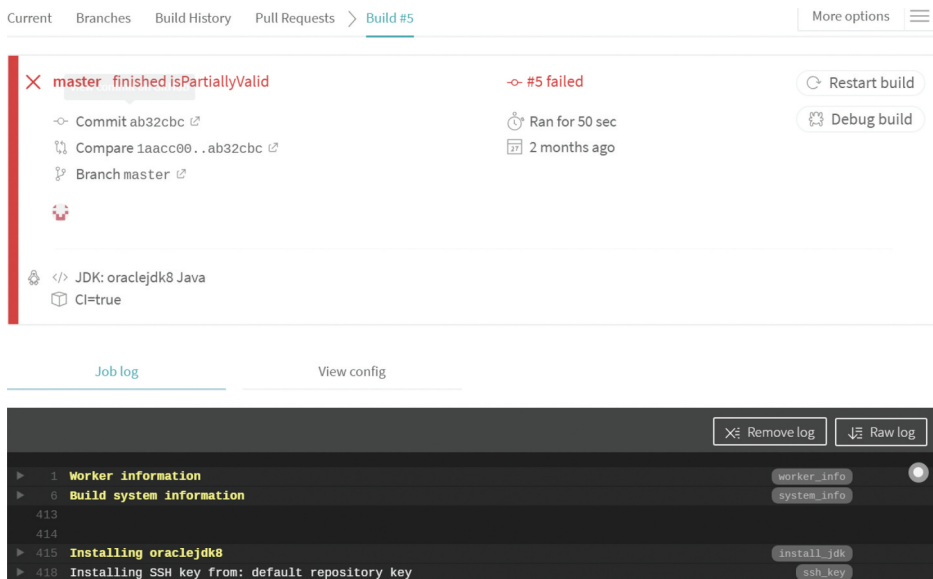
The combination of GitHub, Gradle, and Travis-CI is capable of providing students automated formative feedback without being obtrusive to their learning experience. Nearly all tested automated feedback systems belong to pull technologies (e.g., Web-CAT, AutoLab) (Kendall & Kendall, 1999). Such systems are not an integral part of the coding, debugging, and testing process when students work on programming assignments (Parihar et al., 2017). Students need to submit their code manually every time if they seek automated feedback. If students fail to utilize such systems until it is too late, it will substantially reduce the effectiveness of formative feedback, or even become simply summative (Hao et al., 2019b). Although IDE plugins do exist for platforms such as Web-CAT they lack the utility of Git and force students to use the tools with which they are compatible. In contrast, the combination of GitHub, Gradle, and Travis-CI belongs to push technologies (Kendall & Kendall, 1999). Git, as a version control system adopted by GitHub, is an integral part of students' coding process. When students push their code



to GitHub, it will trigger Travis-CI to test the pushed code and push formative feedback to students. The architecture of the automated feedback system based on GitHub, Gradle, and Travis-CI is presented in Figure 1. The feedback associated with each push will be displayed and archived on the Travis-CI website (see Figure 2).



**Figure 1.** The architecture of automated feedback system based on GitHub and Travis-CI.



**Figure 2.** A partial screenshot of what a student sees on the Travis-CI website.



### 3.3. Experiment design

To answer the first two research questions, we conducted a controlled experiment on 76 students taking a CS2 course in a large university in the Pacific Northwest of the United States. The course (and the study) ran for 10 weeks, and is composed of three 50-min lectures and two 90-min lab sections per week.

Each student was required to finish three complex individual programming assignments during the lab sections. Each programming assignment required about 50 to 300 lines of code to complete. Each assignment was designed to have students complete a set of functions and apply these functions in the main program. Automated feedback was provided on both the individual functions implemented by students and the main program with randomized input data (see Figure 3 for an example). Student performance on each assignment is determined by the percentage of test cases that are passed.

Setting up different types of formative feedback is essential to answer the first two research questions. Given the same unit test, we prepared a collection of automated feedback based on different input data by using both the representative edge cases and randomly generated data. In terms of the designs of the feedback, we adopted the feedback classification of Narciss and Huth (Narciss & Huth, 2004), developing three types of feedback per unit test: knowledge of results (KR), knowledge of correct responses (KCR), and elaborated feedback (EF).

To form groups for the controlled experiment, we took advantage of the setups of the pre-existing lab sections. Before the course started, students enrolled in this course were randomly assigned to one of the three different lab sections which they exclusively attended for the entire semester. Each lab section was facilitated by a teaching assistant; all teaching assistants went through the same training before the course. This setup

```

350 StringManipulatorTest > testReplaceAll2 FAILED
351 java.lang.AssertionError: This is a test on your replaceAll method.
352 The original string is: catch that banana.
353 When c gets replaced by d
354 Expected: is "datdh that banana"
355 but: was ""
356 at org.hamcrest.MatcherAssert.assertThat(MatcherAssert.java:20)
357 at org.junit.Assert.assertThat(Assert.java:956)
358 at StringManipulatorTest.testReplaceAll2(StringManipulatorTest.java:22)
359
360 StringManipulatorTest > testReplaceSingle2 FAILED
361 java.lang.AssertionError: This is a test on your validateOccurrence method.
362 The original string is: good morning.
363 When the character at index 11 gets replaced by d
364 Expected: is "good mornind"
365 but: was ""
366 at org.hamcrest.MatcherAssert.assertThat(MatcherAssert.java:20)
367 at org.junit.Assert.assertThat(Assert.java:956)
368 at StringManipulatorTest.testReplaceSingle2(StringManipulatorTest.java:69)
369
370 StringManipulatorTest > testRemove FAILED
371 java.lang.AssertionError: This is a test on your remove method.
372 The original string is: good morning.
373 When the character o gets removed
374 Expected: is "gd mrning"
375 but: was ""
376 at org.hamcrest.MatcherAssert.assertThat(MatcherAssert.java:20)
377 at org.junit.Assert.assertThat(Assert.java:956)
378 at StringManipulatorTest.testRemove(StringManipulatorTest.java:78)
379
380 StringManipulatorTest > testValidateOccurrence FAILED
381 java.lang.AssertionError: This is a test on your validateOccurrence method.

```

**Feedback on test 1**

**Feedback on test 2**

**Feedback on test 3**

**Feedback on test 4**

Figure 3. An example of feedback from multiple unit testing.

helped minimize possible communication across lab sections as the student body for each section was consistent throughout the quarter. Therefore, we treated students in a lab session as a group and provided each group with different types of automated feedback:

- Group KR (25 students): *KR*
- Group KCR (25 students): *KR + KCR*
- Group EF (26 students): *KR + KCR + EF*

The *KR* was implemented as the information indicating if a test case passed or failed. The *KCR* feedback was implemented as the input and contrasts between the expected output and actual output. The *EF* feedback was implemented as a one-level hint that addresses the top five mistakes summarized from student code submissions in the prior three quarters. If the mistake can be mapped to a misconception, the hint would be coupled with more explanations that addressed the misconception. We loosely followed the guidelines of Haldeman et al. (2018) in our implementation of the *EF* feedback. An example of the feedback each group received on the same function is presented in Figure 4.

It is worth noting that we did not set up a group receiving no automated feedback or a group receiving the combination of *KR* and *EF*. There is abundant evidence that providing some feedback is always more beneficial to student learning as long as the feedback does not confuse students further (Higgins et al., 2002; Juwah et al., 2004; Parihar et al., 2017). Without *KCR* feedback, the combination of *KR* and *EF* may make less sense to students, and the effects of *EF* might be substantially reduced in this case. Without knowing the gap between the expected and actual results, hints can do little to direct students in the right direction. The ultimate goal of this study is to understand what feedback has the greatest impact on student learning, and use this understanding to

```

584 StringManipulatorTest > testReplaceSingle FAILED
585     java.lang.AssertionError:
586     This is the test on your replaceSingle method.
587     You failed to pass this test.
                                     (Group KR)
432 StringManipulatorTest > testReplaceSingle FAILED
433     java.lang.AssertionError:
434     This is the test on your replaceSingle method.
435     You failed to pass the test.
436     The original string is: good morning
437     When the character at index 3 gets replaced by p
438     Expected: is "goop morning"
439     but: was ""
                                     (Group KCR)
584 StringManipulatorTest > testReplaceSingle FAILED
585     java.lang.AssertionError:
586     This is the test on your replaceSingle method.
587     You failed to pass the test.
588     Hint: You may forget to update the target string to return, as it is still empty.
589     For instance, the original string is: good morning
590     When the character at index 3 gets replaced by p
591     Expected: is "goop morning"
592     but: was ""
                                     (Group EF)

```

Figure 4. An example of different feedback configurations for a method.

guide the design and improvement of automated feedback systems. The possible group configurations, such as receiving no automated feedback or receiving the combination of *KR* and *EF* are not the best options to help achieve the goal. As a result, we did not include the two possible groups in our research design.

### 3.4. Data collection

To answer the first research question “How do different types of automated formative feedback impact student performance on programming assignments?”, we collected student performance on all three programming assignments.

In answering the second research question “How do different types of automated formative feedback impact student interaction with the automated feedback system?”, we collected the following student behavior data through APIs of GitHub and Travis-CI:

- *Timestamps*: Timestamps of pushes per student per assignment.
- *Efforts*: Number of lines changed since the last push per student per assignment.
- *Automated testing results*: Which test cases are passed per push per student per assignment.

Finally, to answer the third research question “How do students perceive different types of automated formative feedback?”, we surveyed all students by the end of the course using four questions:

- How often did you use the feedback from Travis-CI?
- What do you do when you find you have failed a test case on Travis-CI? Describe your experience of utilizing the feedback from Travis-CI.
- What do you like about the automated feedback from Travis-CI?
- What do you dislike about the automated feedback from Travis-CI? How do you want us to improve it?

## 4. Results

### 4.1. How do different types of automated formative feedback impact student performance on programming assignments?

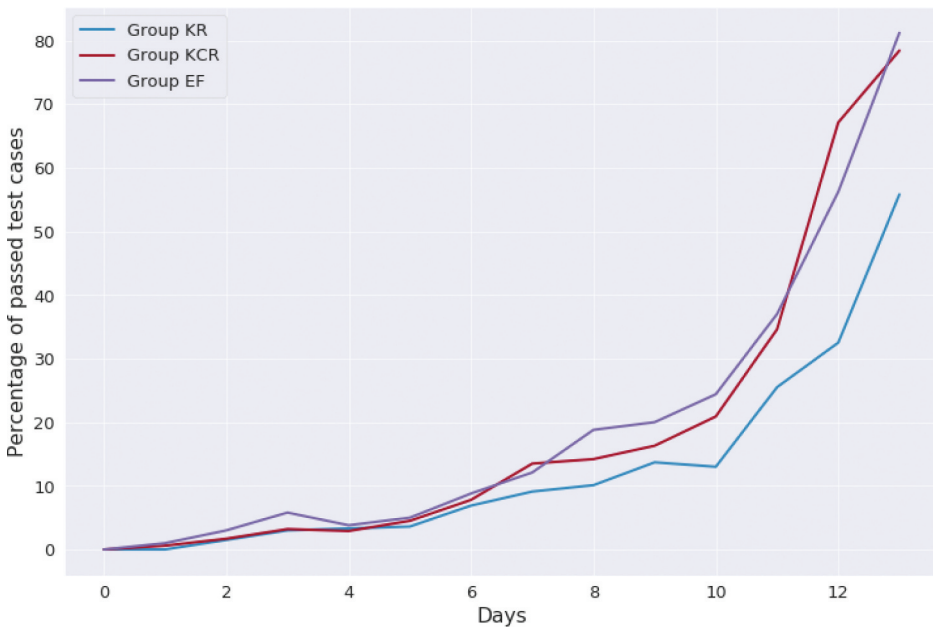
We summarized the performance comparisons among the three groups’ programming assignment in [Table 1](#). Students in Group *EF* seem to have better performance than in Group *KCR* and *KR*, while students in Group *KCR* seem to have better performance than in Group *KR*.

To gain insights into this group performance differences over time, we plotted student progress over time by group. Student progress can be evidenced by the average percentage of passed test cases. For instance, when we plotted student progress of Assignment 2

**Table 1.** Average student performance by group and by assignment.

	Assignment Averages		
	Assignment 1	Assignment 2	Assignment 3
Group <i>KR</i>	75.30	77.45	67.50
Group <i>KCR</i>	91.75	85.59	84.29
Group <i>EF</i>	92.44	89.65	90.65

The total of each assignment is 100.



**Figure 5.** Percentage of passed test cases by group on Assignment 2.

by group, we found that Group KR tended to make progress more slowly than their counterparts in Group KCR and EF, and did not pass as many test cases as Group KCR and EF by the deadline either (see Figure 5). A similar pattern was found in Assignment 1 and 3.

To quantify the observed pattern, we performed One-Way Multiple Analysis of Variance (MANOVA) to examine the student performance differences on the three assignments across three groups. To address the possible issue of bimodal performance in CS courses, we performed log transformation on the data of student assignment performance first. The Shapiro–Wilk test yielded a non-significant result ( $p > 0.05$ ), indicating that the transformed data is normally distributed. The Variance Inflation Factors (VIF) of student performance of the three assignments were all smaller than 2.5, indicating a mild correlation among student performance across assignments. Box’s test yielded a non-significant result ( $p = 0.19$ ), indicating that the homogeneity of variance-covariance matrices could be assumed.

The results of MANOVA indicated a significant difference [Wilk’s  $\Lambda = 0.45$ ,  $F(2, 73) = 9.12$ , and  $p < 0.001$ ]. To understand which two groups had significantly different performance, we followed up MANOVA with discriminant analysis, which revealed two discriminant functions. The first function explained 95.0% of the variance, canonical  $R^2 = 0.69$ , whereas the second explained 2.0%, canonical  $R^2 = 0.15$ . In combination, these discriminative functions significantly differentiated Group KR from Group KCR/EF [ $\Lambda = 0.43$ ,  $\chi^2 = 44.17$ ,  $p < 0.001$ ], but removing the first function indicated that the second function did not significantly differentiate the remaining two groups [ $\Lambda = 0.91$ ,  $\chi^2 = 1.27$ ,  $p > 0.5$ ]. In summary, the significant differences detected by MANOVA only existed between Group KR and Group KCR/EF.

#### 4.2. How do different types of automated formative feedback impact student interaction with the automated feedback system?

To answer the second research question, we studied (a) student efforts on each programming assignment over time and (b) the behaviors of seeking automated feedback by group. Student efforts were evidenced by the number of changed lines of code on a daily basis. When aggregated, student efforts per assignment showed largely the same pattern. Students tended to make little to moderate efforts in the early stage after an assignment was given, stagnate in the middle stage, and make a tremendous amount of effort towards the due dates. However, we did not see any obvious differences across groups in terms of student efforts (see Figure 6). We followed up this observation with MANOVA on averaged student efforts but did not find statistically significant results either [ $F(2, 73) = 1.35$ , and  $p > 0.05$ ].

Student feedback-seeking behaviors were inferred from student actions of pushing their code to GitHub, which will trigger the automated testing and feedback to be sent to students. When aggregated, student feedback-seeking behaviors showed inconsistent patterns across the assignments. Students from Group KR seemed to seek more feedback than their counterparts in Group KCR and EF on the last two assignments, but a similar amount of feedback on the first assignment (see Figure 7).

We followed up the above observation with MANOVA to quantify the differences in student feedback-seeking behaviors. The results of MANOVA indicated a significant difference [Wilk's  $\Lambda = 0.37$ ,  $F(2, 73) = 5.36$ , and  $p < 0.05$ ]. Discriminant analysis was used to determine where the difference was. Two discriminant functions were built. The first function explained 91.0% of the variance, canonical  $R^2 = 0.16$ , whereas the second explained 6.0%, canonical  $R^2 = 0.01$ . In combination, these discriminative functions significantly differentiated Group KR from Group KCR/EF [ $\Lambda = 0.89$ ,  $\chi^2 = 21.29$ ,  $p < 0.05$ ], but removing the first function indicated that the second function did not significantly differentiate the remaining two groups [ $\Lambda = 0.96$ ,  $\chi^2 = 1.31$ ,  $p > 0.05$ ]. In summary, the significant differences detected by MANOVA only existed between Group KR and Group KCR/EF.

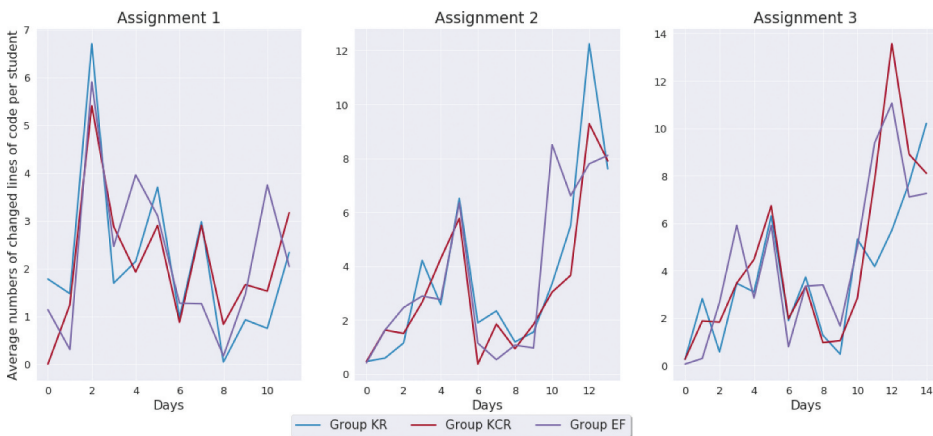
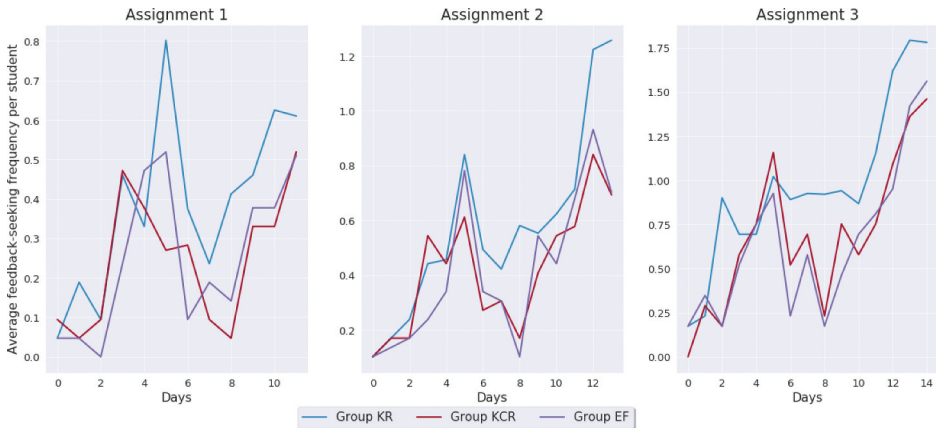


Figure 6. Number of changed lines of code by group per assignment.



**Figure 7.** Feedback-seeking frequencies by group per assignment.

### 4.3. How do students perceive different types of automated formative feedback?

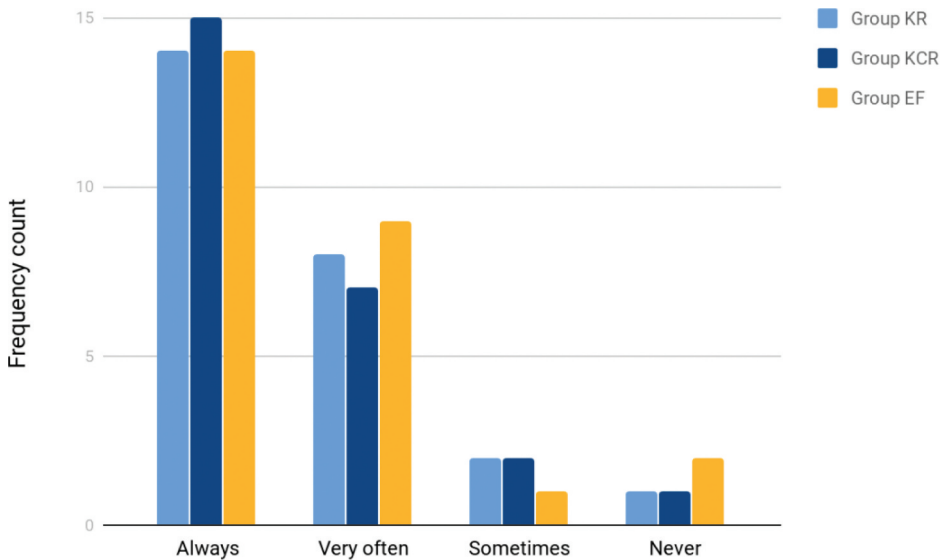
To understand student perception of automated formative feedback, we analyzed student answers to the three survey questions. Question 1 adopted a four-point Likert scale to assess the degree to which students utilized the automated formative feedback pushed from Travis-CI. We used the chi-square test to compare student answers to this question across groups. Questions 2 to 4 were open-ended. We applied grounded theory analysis to the answers to the three questions, starting with open coding, moving to axial coding sequentially (Corbin & Strauss, 2008). We summarized the analysis results and identified themes in the following sections.

#### 4.3.1. Question 1: how often did you use the feedback from Travis-CI?

For the first question, we used a four-point Likert scale to assess the degree to which students utilized the automated formative feedback pushed from Travis-CI. The responses from students are summarized in Figure 8. Overall, students across all groups reported that they utilized formative feedback frequently. Although students from Group EF were on the higher end of the scale, we did not find any significant differences between groups using the Chi-squared test [ $\chi^2(6, 73) = 2.89, p > 0.05$ ].

#### 4.3.2. Question 2: What do you do when you find you have failed a test case on Travis-CI? Describe your experience of utilizing the feedback from Travis-CI

We identified a single theme with five steps in terms of how students utilize automated formative feedback from student responses across the three groups: (1) Analyze Travis's output and attempt to understand the cause of the error, (2) Identify the location within the source code responsible for the error, (3) Attempt a fix, (4) Push changes to Travis, and (5) Repeat until all test cases are passed. It is worth noting that many students mentioned that they would reference outside sources such as StackOverflow (stackoverflow.com) to help understand the feedback if there was a bug in their code. Several students from Group KR mentioned that they had a difficult time figuring out why they failed a test case or needed to develop their own test cases, but no students from Group KCR or EF mentioned the same experience.



**Figure 8.** Frequency of utilizing automated formative feedback by group.

#### **4.3.3. Question 3: what do you like about the automated feedback from Travis-CI?**

We identified two major themes from student responses across the three groups, including the positive psychological effects and unobtrusiveness of the feedback system to their learning experience. On the unobtrusiveness of the feedback system, many students mentioned that they felt that they were with a tutor that could tell them what went wrong constantly without interfering with their learning experience. One student mentioned that “the system creates a cohesive and supportive environment ...”. On the positive psychological effects, many students commented on the satisfaction they had when automated feedback indicated that they had passed all test cases. One student said “... when I turn in a lab on Travis it gives me peace of mind that I’ll get a good grade on the assignment ...”. Another student stated that “It felt good to see the green light on Travis (that indicates all test cases are passed) ...”.

#### **4.3.4. Question 4: What do you dislike about the automated feedback from Travis-CI? How do you want us to improve it?**

We identified two major themes and a notable difference between Group KR and Group KCR/EF. The two themes include the speed of the system and the capability of the system to allow users to add test cases. On the capability of the system to allow users to add test cases, many students expressed that they wished the system could take the test cases they wanted to add.

On the speed of the system, many students expressed that sometimes it took too long (up to tens of minutes) for them to get the automated feedback (after they pushed their code to GitHub). Despite the numerous advantages of the implemented system (e.g., cost-free, replicability), it is running on one single-threaded build service that will queue all tests that need to be performed. We observed that students tended to procrastinate in the beginning, and did not start committing more time and effort to work on the



assignments when due dates were approaching. This will end up with many testing tasks queued by the system, and delay the system from pushing feedback to all students from instantly to tens of minutes.

The major difference we identified between groups is the difficulty in interpreting the automated feedback. Many students from Group KR complained about the difficulties in interpreting the feedback. One student commented that "...I know that I failed a test case, but I don't know where I was wrong ...". In contrast, we can not find a similar complaint in the answers from Group KCR and EF.

## 5. Discussion

### 5.1. Impacts of different types of feedback on student learning

We investigated the degree of differential performance that exists between three groups, each provided with one of the following feedback categories:

(1) *Knowledge of Results (KR)*: feedback includes information on what test cases failed.

(2) *Knowledge of Correct Responses (KCR)*: KR + Additional information on where in the test case the failure occurred.

(3) *Elaborated Feedback (EF)*: KCR + A hint regarding what the student might need to do to fix their code.

Among all the findings, we would like to highlight two results of the impacts of different types of feedback on student learning and student perception towards the feedback.

Across three complex programming assignments, we found that students who received higher levels of feedback (*KCR* and *EF*) outperformed their counterparts who received only *KR* feedback significantly. This result is consistent with research on feedback design in other contexts (Hattie & Timperley, 2007; Van der Kleij et al., 2012), and partially confirmed the findings of Hao et al. (2019b). Additionally, we found that students receiving only *KR* feedback tended to complain more about the difficulties in interpreting the feedback, and seek feedback more frequently than students receiving higher levels of feedback. At first glance, it may seem counterintuitive that the group that complained the most about feedback sought feedback most frequently. Early studies on this topic raised concerns of students abusing automated feedback over similar observations (Cheang et al., 2003; Chen, 2004; Guerreiro & Georgouli, 2006). Without questioning the efficacy of the feedback, we may reach a similar conclusion. However, when we put these results together with the evidence on student performance differences, it helps us gain a deep understanding of the efficacy of feedback on correctness. A more reasonable interpretation is that the low interpretability of *KR* feedback leads to difficulty in effective debugging, which in turn leads to more trial and error behaviors in testing.

We found insignificant performance differences between students receiving *KCR* feedback and students receiving both *KCR* and *EF* feedback. There are two possible interpretations for this result:

- *KCR* feedback is sufficient for most cases: The comparison between expected and actual outputs can inform students of both where they are currently and where the destination is (Nicol & Macfarlane-Dick, 2006; Sadler, 1989). With these two pieces of information, students are able to figure out how to get to the destination from their current positions. If this is true,

hints and more detailed explanations might not be needed for every problem students encounter. Providing hints and more detailed explanations that effectively address learning misconceptions could be either computationally expensive or require great attention from human instructors (Hao et al., 2019b). If some bottlenecks or the most challenging components of a given problem can be identified as worthy of hints and detailed explanations, it will be more efficient in terms of either computational power or instructors' time. Additionally, providing *KCR* feedback without hints may strengthen student debugging skills because conquering the problem means that students have figure out "how to get the destination from where they are currently" (Nicol & Macfarlane-Dick, 2006). These are questions that need to be answered by future studies on this topic.

- *EF* feedback needs further investigation: We loosely followed the guidelines of Haldeman et al. (2018) in our implementation of the *EF* feedback, and only implemented the hints for the top five most common mistakes. On the one hand, we found that one-level hints were not significantly beneficial to student learning. On the other hand, students did not demonstrate any evidence of abusing one-level hints. One-level hints do not risk revealing the direct answers to students, but may also not be adaptive enough to help students overcome the difficulties they meet. This may explain the findings of this study. Multi-leveled hints may be more effective in helping students overcome difficulties, but may also lead students to game the system by constantly accessing hints at deeper levels without a sincere effort in problem-solving (Aleven et al., 2016; D Baker et al., 2006; Baker et al., 2008). Future studies may consider comparing the efficacy of different designs on hints through controlled experiments.

We do not yet have the evidence to distinguish which of these interpretations is a better account for the results. However, we believe that more investigations to understand the feedback design of effective automated feedback are necessary.

## **5.2. Feedback designs for automated testing and feedback systems**

We have established that there is a necessity to understand the effects of different designs of feedback, and use this knowledge to guide the design of automated testing and feedback systems. Developing a stand-alone system that manages student programming assignments and grades, tests student code, and provides feedback is challenging and also requires a significant amount of maintenance effort. Therefore, many of the automated feedback systems reported in prior studies are no longer accessible to the public. Future development efforts may consider taking advantage of the existing services that are popular in programming courses (e.g., GitHub) and adopting the paradigm of micro-services (Hao & Tsikerdekis, 2019). As such, the development efforts could solely focus on automated testing and feedback.

Effective feedback design may need to find a balance between providing adaptive help and being cost-effective. The findings of this study can provide guidance on this challenge. Simple feedback can be effective. However, to realize its full potential, the information of feedback should not be restricted to the extent of being difficult to interpret. Standing alone, *KR* feedback is not very helpful to student learning. However, *KR* and *KCR* feedback, when combined, can be positively impactful. Based on the findings of this study, we would recommend that the minimum amount of information automated feedback covers to include both correctness and the gap between expected and actual outputs.

Despite of the recent advances in automatic generation of adaptive feedback, it is still extremely challenging to generate hints and detailed explanations that address student

learning misconceptions automatically. The techniques of automatic feedback generation typically require a massive amount of student code submissions, which further limits their application in authentic educational settings (Gulwani et al., 2018; Wang et al., 2018). The findings of our study provide practical guidelines for the scenario where automatic generation of hints and detailed explanations is too difficult or not possible – *KR* and *KCR* feedback, without hints and detailed explanations, can still help students sufficiently on many components of problem-solving.

The last point that should be addressed is the student system gaming behavior. Prior studies on this topic tended to focus on discouraging student gaming behaviors by upgrading the feedback systems, or understanding student motivations to game feedback systems (Baker et al., 2008). Few studies studied the design of feedback itself in the context of computing education. Restricting the information delivered by the feedback is believed as one conventional approach to discourage system gaming behaviors. However, we found that when feedback was very restrictive, it may incite trial and error behaviors. In contrast, we also observed that students did not abuse one-level hints. Many prior studies observed that multi-leveled feedback tended to be abused by students, especially when the low-level feedback was very close to direct answers. A fair question that future studies can ask is whether the design of feedback delivery may cause student gaming behaviors. The understanding of whether and how feedback design contributes to system gaming behaviors will help us focus on the essential development of effective feedback systems.

## 6. Limitations

Our study is not without limitations. First, this study was conducted as a quasi-controlled experiment in only one higher education institute with 76 participants. Although this number is sufficient for a three-group controlled experiment, the sample size is still comparatively small. It is also unclear whether the findings can be scaled to a bigger population. To further verify the generalizability of our findings, future studies may consider exploring the possibility of randomized controlled experiments, and replicating our studies across multiple higher education institutes. Second, our implementation of *KCR* feedback might not be a direct interpretation of the concept of *KCR* proposed by Narciss and Huth (2004). It is natural to interpret *KCR* feedback for a unit testing as the correct solution, or the functioning code of the required method. We did not provide such as the *KCR* feedback. Instead, we provided the input and the contrasts between the actual and expected output. Although our implementation allows room for students to make improvements and try again based on the *KCR* feedback, it may not strictly adhere to the feedback classification system proposed by Narciss and Huth (2004). Third, some factors were not controlled or could not be strictly controlled. Although the teaching assistants that facilitate lab sections went through the same training, their helpfulness and efficiency were not measured or compared. Students' prior knowledge of the subject was not measured, and their demographic information was not collected. The lack of such data prevents the control of noises and further investigation into how students of different gender, race, or age interact with automated feedback systems differently. Future studies may consider collecting such data through surveys and aptitude tests (e.g., Smith IV et al., 2019; Tukiainen & Mönkkönen, 2002), and provide a more fine-grained analysis of how

students' prior knowledge and demographics influence their interactions with automated feedback, especially automated feedback of different designs. Fourth, we performed log transformation on student data to meet the expected assumptions of the adopted statistical analysis, which risks lessening the interpretativity of the effective sizes. Last, how different designs of automated formative feedback impact student knowledge transfer was not investigated in this study. Effective knowledge transfer is the ultimate goal of learning and teaching. Future studies may consider investigating the impacts of designs of automated feedback on student knowledge transfer, building on top of a series of validity assessment of programming assignments and exams.

## 7. Conclusion

This study investigated the impacts of feedback design on the efficacy of automated feedback and how students interact with and perceive different feedback designs. The results revealed that feedback on correctness, if provided as the only type of feedback, may not be helpful to student learning, and may stimulate trial and error debugging behaviors. In contrast, feedback that addresses comparisons between expected and actual outputs was found significantly impactful on student learning, and sufficient for students to move forward for most problems. The results of this study contribute to a fine-grained understanding of feedback design, and provide insights into how we can improve the design of automated testing and feedback systems for programming assignments.

## Disclosure statement

No potential conflict of interest was reported by the authors.

## Notes on contributors

**Qiang Hao** is an assistant professor of computer science & science education at Western Washington University.

**David H. Smith IV** is a Ph.D. student at University of Illinois at Urbana-Champaign.

**Lu Ding** is an instructional designer at Eastern Illinois University.

**Amy Ko** is a professor of informatics at University of Washington Seattle.

**Camille Ottaway** is a computer science teacher at DSST Public schools in Denver, Colorado. Also, Camille Ottaway is a research student that worked with Dr. Qiang Hao at Western Washington University.

**Jack Wilson** is a research student that worked with Dr. Qiang Hao at Western Washington University.

**Kai Arakawa** is a research student that worked with Dr. Qiang Hao at Western Washington University.

**Alistair Turcan** is a research student that worked with Dr. Qiang Hao at Western Washington University.

**Timothy Poehlman** is a research student that worked with Dr. Qiang Hao at Western Washington University.

**Tyler Greer** is a research student that worked with Dr. Qiang Hao at Western Washington University.

## ORCID

Qiang Hao  <http://orcid.org/0000-0001-6361-5035>

## References

- Ala-Mutka, K. M. (2005). A survey of automated assessment approaches for programming assignments. *Computer Science Education, 15*(2), 83–102. <https://doi.org/10.1080/08993400500150747>
- Alemán, J. L. F. (2010). Automated assessment in a programming tools course. *IEEE Transactions on Education, 54*(4), 576–581. <https://doi.org/10.1109/TE.2010.2098442>
- Aleven, V., Roll, I., McLaren, B. M., & Koedinger, K. R. (2016). Help helps, but only so much: Research on help seeking with intelligent tutoring systems. *International Journal of Artificial Intelligence in Education, 26*(1), 205–223. <https://doi.org/10.1007/s40593-015-0089-1>
- Alvarez, I., Espasa, A., & Guasch, T. (2012). The value of feedback in improving collaborative writing assignments in an online learning environment. *Studies in Higher Education, 37*(4), 387–400. <https://doi.org/10.1080/03075079.2010.510182>
- Baker, R., Walonoski, J., Heffernan, N., Roll, I., Corbett, A., & Koedinger, K. (2008). Why students engage in “gaming the system” behavior in interactive learning environments. *Journal of Interactive Learning Research, 19*(2), 185–224.
- Bangert-Drowns, R. L., Kulik, C.-L. C., Kulik, J. A., & Morgan, M. (1991). The instructional effect of feedback in test-like events. *Review of Educational Research, 61*(2), 213–238. <https://doi.org/10.3102/00346543061002213>
- Becker, B. A., Denny, P., Pettit, R., Bouchard, D., Bouvier, D. J., Harrington, B.,... Kamil, A., Karkare, A., McDonald, C., Osera, P. M., & Pearce J. L. (2019). Unexpected tokens: A review of programming error messages and design guidelines for the future. In *Proceedings of the 2019 ACM Conference on Innovation and Technology in Computer Science Education* (pp. 253–254).
- Black, P., & William, D. (2009). Developing the theory of formative assessment. *Educational Assessment, Evaluation and Accountability (Formerly: Journal of Personnel Evaluation in Education), 21*(1), 5. <https://doi.org/10.1007/s11092-008-9068-5>
- Boud, D., & Molloy, E. (2013). *Feedback in higher and professional education: Understanding it and doing it well*. Routledge.
- Brinko, K. T. (1993). The practice of giving feedback to improve teaching: What is effective? *The Journal of Higher Education, 64*(5), 574–593. <https://doi.org/10.1080/00221546.1993.11778449>
- Butler, D. L., & Winne, P. H. (1995). Feedback and self-regulated learning: A theoretical synthesis. *Review of Educational Research, 65*(3), 245–281. <https://doi.org/10.3102/00346543065003245>
- Camp, T., Adrion, W. R., Bizot, B., Davidson, S., Hall, M., Hambrusch, S., Walker, E., & Zweben, S. (2017). Generation cs: The growth of computer science. *ACM Inroads, 8*(2), 44–50. <https://doi.org/10.1145/3084362>
- Cassel, L., & Fox, E. (2000). *Acm journal of education resources in computing*. ACM.
- Cheang, B., Kurnia, A., Lim, A., & Oon, W.-C. (2003). On automated grading of programming assignments in an academic institution. *Computers & Education, 41*(2), 121–131. [https://doi.org/10.1016/S0360-1315\(03\)00030-7](https://doi.org/10.1016/S0360-1315(03)00030-7)
- Chen, P. M. (2004). An automated feedback system for computer organization projects. *IEEE Transactions on Education, 47*(2), 232–240. <https://doi.org/10.1109/TE.2004.825220>
- Chow, S., Yacef, K., Koprinska, I., & Curran, J. (2017). Automated data-driven hints for computer programming students. In *Adjunct Publication of the 25th Conference on User Modeling, Adaptation and Personalization*, ACM, 5–10.
- Computing Research Association (2017). *Generation cs: Computer science undergraduate enrollments surge since 2006*.
- Corbin, J. M., & Strauss, A. L. (2008). *Basics of qualitative research: Techniques and procedures for developing grounded theory*. SAGE Publications, Inc.

- D Baker, R. S., Corbett, A. T., Koedinger, K. R., & Roll, I. (2006). Generalizing detection of gaming the system across a tutoring curriculum. In *International conference on intelligent tutoring systems*, Springer, 402–411.
- Daly, C., & Horgan, J. M. (2004). An automated learning system for java programming. *IEEE Transactions on Education*, 47(1), 10–17. <https://doi.org/10.1109/TE.2003.816064>
- Drummond, A., Lu, Y., Chaudhuri, S., Jermaine, C., Warren, J., & Rixner, S. (2014). Learning to grade student programs in a massive open online course. In *2014 IEEE International Conference on Data Mining*, IEEE, 785–790.
- Edwards, S. H., & Perez-Quinones, M. A. (2008). Web-cat: Automatically grading programming assignments. In *Proceedings of the 13th Annual Conference on Innovation and Technology in Computer Science Education, ITiCSE '08*, ACM, 328.
- Evans, C. (2013). Making sense of assessment feedback in higher education. *Review of Educational Research*, 83(1), 70–120. <https://doi.org/10.3102/0034654312474350>
- Fyfe, E. R., & Rittle-Johnson, B. (2016). Feedback both helps and hinders learning: The causal role of prior knowledge. *Journal of Educational Psychology*, 108(1), 82. <https://doi.org/10.1037/edu0000053>
- Gao, J., Pang, B., & Lumetta, S. S. (2016). Automated feedback framework for introductory programming courses. In *Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education*, ACM, 53–58.
- Gielen, S., Peeters, E., Dochy, F., Onghena, P., & Struyven, K. (2010). Improving the effectiveness of peer feedback for learning. *Learning and Instruction*, 20(4), 304–315. <https://doi.org/10.1016/j.learninstruc.2009.08.007>
- Greer, T., Hao, Q., Jing, M., & Barnes, B. (2019, February). On the effects of active learning environments in computing education. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education* (pp. 267–272).
- Guerreiro, P., & Georgouli, K. (2006, June). Combating anonymousness in populous CS1 and CS2 courses. In *Proceedings of the 11th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education, ITiCSE '06*, (pp. 8–12).
- Gulwani, S., Radiček, I., & Zuleger, F. (2018). Automated clustering and program repair for introductory programming assignments. In *Proceedings of the 39th ACM SIGPLAN Conference on Programming Language Design and Implementation*, ACM, 465–480.
- Gusukuma, L., Bart, A. C., & Kafura, D. (2020, February). Pedal: An infrastructure for automated feedback systems. In *Proceedings of the 51st ACM Technical Symposium on Computer Science Education*, 1061–1067.
- Haldeman, G., Tjang, A., Babeș-Vroman, M., Bartos, S., Shah, J., Yucht, D., & Nguyen, T. D. (2018). Providing meaningful feedback for autograding of programming assignments. In *Proceedings of the 49th ACM Technical Symposium on Computer Science Education*, ACM (pp. 278–283).
- Hao, Q., Smith IV, D. H., Iriumi, N., Tsikerdekis, M., & Ko, A. J. (2019a). A systematic investigation of replications in computing education research. *ACM Transactions on Computing Education (TOCE)*, 19(4), 1–18. <https://doi.org/10.1145/3345328>
- Hao, Q., & Tsikerdekis, M. (2019). How automated feedback is delivered matters: Formative feedback and knowledge transfer. In *2019 IEEE Frontiers in Education Conference (FIE)*, IEEE, 1–6.
- Hao, Q., Wilson, J. P., Ottaway, C., Iriumi, N., Arakawa, K., Smith, I., & David, H. (2019b). Investigating the essential of meaningful automated formative feedback for programming assignments. In *2019 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC'19)*, IEEE, 1–5.
- Hao, Q., Wright, E., Barnes, B., & Branch, R. M. (2016). What are the most important predictors of computer science students' online help-seeking behaviors? *Computers in Human Behavior*, 62, 467–474. <https://doi.org/10.1016/j.chb.2016.04.016>
- Hattie, J., & Gan, M. (2011). Instruction based on feedback. In Mayer, R. E., & Alexander, P. A. (Eds.). *Handbook of research on learning and instruction* (pp. 263–285). Taylor & Francis.
- Hattie, J., & Timperley, H. (2007). The power of feedback. *Review of Educational Research*, 77(1), 81–112. <https://doi.org/10.3102/003465430298487>



- Head, A., Glassman, E., Soares, G., Suzuki, R., Figueredo, L., D'Antoni, L., & Hartmann, B. (2017). Writing reusable code feedback at scale with mixed-initiative program synthesis. In *Proceedings of the Fourth (2017) ACM Conference on Learning@ Scale*, ACM, 89–98.
- Higgins, R., Hartley, P., & Skelton, A. (2002). The conscientious consumer: Reconsidering the role of assessment feedback in student learning. *Studies in Higher Education*, 27(1), 53–64. <https://doi.org/10.1080/03075070120099368>
- Hundhausen, C. D., & Brown, J. L. (2007). An experimental study of the impact of visual semantic feedback on novice programming. *Journal of Visual Languages & Computing*, 18(6), 537–559. <https://doi.org/10.1016/j.jvlc.2006.09.001>
- Ihantola, P., Ahoniemi, T., Karavirta, V., & Seppälä, O. (2010). Review of recent systems for automatic assessment of programming assignments. In *Proceedings of the 10th Koli calling international conference on computing education research*, ACM, 86–93.
- Ihantola, P., Vihavainen, A., Ahadi, A., Butler, M., Börstler, J., Edwards, S. H., Isohanni, E., Korhonen, A., Petersen, A., Rivers, K., ... & Rubio MÁ. (2015). Educational data mining and learning analytics in programming: Literature review and case studies. In *Proceedings of the 2015 ITICSE on Working Group Reports* (pp. 41–63).
- Jawah, C., Macfarlane-Dick, D., Matthew, B., Nicol, D., Ross, D., & Smith, B. (2004). Enhancing student learning through effective formative feedback. *The Higher Education Academy*, 140, 1–40.
- Karvelas, I., Li, A., & Becker, B. A. (2020, February). The effects of compilation mechanisms and error message presentation on novice programmer behavior. In *Proceedings of the 51st ACM Technical Symposium on Computer Science Education* (pp. 759–765).
- Kendall, J. E., & Kendall, K. E. (1999). Information delivery systems: An exploration of web pull and push technologies. *Communications of the Association for Information Systems*, 1(1), 14. <https://doi.org/10.17705/1CAIS.00114>
- Keuning, H., Jeuring, J., & Heeren, B. (2018). A systematic literature review of automated feedback generation for programming exercises. *ACM Transactions on Computing Education (TOCE)*, 19(1), 3. <https://doi.org/10.1145/3231711>
- Kinnunen, P., & Simon, B. (2012). My program is ok—am i? Computing freshmen's experiences of doing programming assignments. *Computer Science Education*, 22(1), 1–28. <https://doi.org/10.1080/08993408.2012.655091>
- Kluger, A. N., & DeNisi, A. (1996). The effects of feedback interventions on performance: A historical review, a meta-analysis, and a preliminary feedback intervention theory. *Psychological Bulletin*, 119(2), 254. <https://doi.org/10.1037/0033-2909.119.2.254>
- Lee, H. W., Lim, K. Y., & Grabowski, B. L. (2010). Improving self-regulation, learning strategy use, and achievement with metacognitive feedback. *Educational Technology Research and Development*, 58(6), 629–648. <https://doi.org/10.1007/s11423-010-9153-6>
- Luxton-Reilly, A., Albluwi, I., Becker, B. A., Giannakos, M., Kumar, A. N., Ott, L., Paterson, J., Scott, M. J., Sheard, J., ... & Szabo, C. (2018, July). Introductory programming: A systematic literature review. In *Proceedings Companion of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education* (pp. 55–106).
- McMillan, J. H., Venable, J. C., & Varier, D. (2013). Studies of the effect of formative assessment on student achievement: So much more is needed. *Practical Assessment, Research, and Evaluation*, 18(1), 2.
- Narciss, S., & Huth, K. (2004). How to design informative tutoring feedback for multimedia learning. In Niegemann, H. M., Leutner, D., Brünken, R., Leutner, D., & Brünken, R. (Eds.), *Instructional design for multimedia learning* (pp 181–195). Waxmann.
- Narciss, S., Sosnovsky, S., Schnaubert, L., Andrès, E., Eichelmann, A., Gogvadze, G., & Melis, E. (2014). Exploring feedback and student characteristics relevant for personalizing feedback strategies. *Computers & Education*, 71, 56–76. <https://doi.org/10.1016/j.compedu.2013.09.011>
- Neve, P., Hunter, G., Livingston, D., & Orwell, J. (2012). Nooblab: An intelligent learning environment for teaching programming. In *2012 IEEE/WIC/ACM International Conferences on Web Intelligence and Intelligent Agent Technology*, 3, IEEE, 357–361.



- Nicol, D. J., & Macfarlane-Dick, D. (2006). Formative assessment and self-regulated learning: A model and seven principles of good feedback practice. *Studies in Higher Education*, 31(2), 199–218. <https://doi.org/10.1080/03075070600572090>
- Nygren, H., Leinonen, J., & Hellas, A. (2019, July). Non-restricted access to model solutions: A good idea? In *Proceedings of the 2019 ACM Conference on Innovation and Technology in Computer Science Education* (pp. 44–50).
- Pardo, A. (2018). A feedback model for data-rich learning experiences. *Assessment & Evaluation in Higher Education*, 43(3), 428–438. <https://doi.org/10.1080/02602938.2017.1356905>
- Parihar, S., Dadachanji, Z., Singh, P. K., Das, R., Karkare, A., & Bhattacharya, A. (2017). Automatic grading and feedback using program repair for introductory programming courses. In *Proceedings of the 2017 ACM Conference on Innovation and Technology in Computer Science Education*, ACM, 92–97.
- Pears, A., Seidman, S., Malmi, L., Mannila, L., Adams, E., Bennedsen, J., Devlin, M., & Paterson, J. (2007). A survey of literature on the teaching of introductory programming. In *ACM sigcse bulletin*, 39, ACM, 204–223.
- Pieterse, V. (2013). Automated assessment of programming assignments. In *Proceedings of the 3rd Computer Science Education Research Conference on Computer Science Education Research*, CSERC '13, Open Univ., Heerlen, The Netherlands, The Netherlands. Open Universiteit, Heerlen, 4: 45–4:56.
- Price, T. W., Zhi, R., Dong, Y., Lytle, N., & Barnes, T. (2018). The impact of data quantity and source on the quality of data-driven hints for programming. In *International Conference on Artificial Intelligence in Education*, Springer, 476–490.
- Sadler, D. R. (1989). Formative assessment and the design of instructional systems. *Instructional Science*, 18(2), 119–144. <https://doi.org/10.1007/BF00117714>
- Saifi, S., Mahmood, T., Gujjar, A., & Ali Sha, S. (2011). Assessing the quality of assessment techniques at higher education level. *International Journal of Business and Social Science*, 2(12), 273.
- Sax, L. J., Lehman, K. J., & Zavala, C. (2017). Examining the enrollment growth: Non-cs majors in cs1 courses. In *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education*, SIGCSE '17, New York, NY, USA, ACM, 513–518.
- Serge, S. R., Priest, H. A., Durlach, P. J., & Johnson, C. I. (2013). The effects of static and adaptive performance feedback in game-based training. *Computers in Human Behavior*, 29(3), 1150–1158. <https://doi.org/10.1016/j.chb.2012.10.007>
- Shute, V. J. (2008). Focus on formative feedback. *Review of Educational Research*, 78(1), 153–189. <https://doi.org/10.3102/0034654307313795>
- Smith IV, D. H., Hao, Q., Dennen, V., Tsikerdekis, M., Barnes, B., Martin, L., & Tresham, N. (2020). Towards understanding online question & answer interactions and their effects on student performance in large-scale stem classes. *International Journal of Educational Technology in Higher Education*, 17(1), 1–15. <https://doi.org/10.1186/s41239-020-00200-7>
- Smith IV, D. H., Hao, Q., Jagodzinski, F., Liu, Y., & Gupta, V. (2019, May). Quantifying the effects of prior knowledge in entry-level programming courses. In *Proceedings of the ACM Conference on Global Computing Education* (pp. 30–36).
- Stobart, G. (2008). *Testing times: The uses and abuses of assessment*. Routledge.
- Tukiainen, M., & Mönkkönen, E. (2002). Programming aptitude testing as a prediction of learning to program. In *Proceedings of 4th Workshop of the Psychology of Programming Interest Group* (pp. 45–57).
- van der Kleij, F. M., Eggen, T. J., Timmers, C. F., & Veldkamp, B. P. (2012). Effects of feedback in a computer-based assessment for learning. *Computers & Education*, 58(1), 263–272. <https://doi.org/10.1016/j.compedu.2011.07.020>
- Van der Kleij, F. M., Feskens, R. C., & Eggen, T. J. (2015). Effects of feedback in a computer-based learning environment on students' learning outcomes: A meta-analysis. *Review of Educational Research*, 85(4), 475–511. <https://doi.org/10.3102/0034654314564881>
- Van Merriënboer, J. J., & Sweller, J. (2005). Cognitive load theory and complex learning: Recent developments and future directions. *Educational Psychology Review*, 17(2), 147–177. <https://doi.org/10.1007/s10648-005-3951-0>

- Vujošević -Janičić, M., Nikolić, M., Tošić, D., & Kuncak, V. (2013). Software verification and graph similarity for automated evaluation of students' assignments. *Information and Software Technology*, 55(6), 1004–1016. <https://doi.org/10.1016/j.infsof.2012.12.005>
- Wang, K., Singh, R., & Su, Z. (2018). Search, align, and repair: Data-driven feedback generation for introductory programming exercises. In *Proceedings of the 39th ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2018*, New York, NY, USA, ACM, 481–495.
- Watson, C., Li, F. W., & Godwin, J. L. (2012). Bluefix: Using crowd-sourced feedback to support programming students in error diagnosis and repair. In *International Conference on Web-Based Learning*, Springer, 228–239.
- Yorke, M. (2001). Formative assessment and its relevance to retention. *Higher Education Research & Development*, 20(2), 115–126. <https://doi.org/10.1080/758483462>
- Yorke, M. (2003). Formative assessment in higher education: Moves towards theory and the enhancement of pedagogic practice. *Higher Education*, 45(4), 477–501. <https://doi.org/10.1023/A:1023967026413>