Investigating the Essential of Meaningful Automated Formative Feedback for Programming Assignments

Qiang Hao[*], Jack P Wilson[†], Camille Ottaway[‡], Naitra Iriumi[§], Kai Arakawa[¶] and David H Smith IV[‖]
Western Washington University
Bellingham, WA, USA
Email: [*]qiang.hao@wwu.edu, [†]wilso313@wwu.edu, [‡]ottawac@wwu.edu,
[§]iriumin@wwu.edu, [¶]hicksk5@wwu.edu, [‖]smithd77@wwu.edu

***Abstract*—This study investigated the essential of meaningful automated feedback for programming assignments. Three different types of feedback were tested, including (a) *What's wrong* - what test cases were testing and which failed, (b) *Gap* - comparisons between expected and actual outputs, and (c) *Hint* - hints on how to fix problems if test cases failed. 46 students taking a CS2 participated in this study. They were divided into three groups, and the feedback configurations for each group were different: (1) Group One - *What's wrong*, (2) Group Two - *What's wrong + Gap*, (3) Group Three - *What's wrong + Gap + Hint*. This study found that simply knowing what failed did not help students sufficiently, and might stimulate system gaming behavior. Hints were not found to be impactful on student performance or their usage of automated feedback. Based on the findings, this study provides practical guidance on the design of automated feedback.**

***Index Terms*—automated feedback, automated grading, formative feedback, programming assignments, computing education, controlled experiments**

I. Introduction

Student interest in computer science (CS) has increased substantially over the last decade. In the U.S., undergraduate CS enrollment has doubled since 2011, and class sizes of programming courses offered in colleges have more than tripled [1]. CS courses nowadays are characterized by large enrollments and low instructor-to-student ratios, especially for the entry-level CS courses, such as CS1, CS2 or data structures [1], [2]. The challenges in assessing programming assignments of a large number of students make it difficult for students to get feedback in time. When students work on either individual or group programming assignments, they may meet challenges they can not overcome. If feedback can be provided at those moments when it is needed the most, the learning efficacy can be significantly enhanced.

Prior studies addressing the feedback challenge originated from automated grading of programming assignments. As class sizes grew rapidly, it was natural to ensure that programming assignments were assessed in a timely manner [3], [4]. The investigation on auto-grading made significant contributions to computing education research, but also cast a perspective of summative feedback on the efforts to automate feedback — feedback should be provided along the assessment results [5], [6]. A popular concern was that formative feedback, the feedback provided during the learning process, may lead to students gaming the system [7]–[9]. Many tested programming assignment systems provide no formative feedback to students or limit the allowed number of submissions [9]–[11]. As a result, there is a gap in our understanding on how students utilize automated formative feedback and whether that leads to better learning efficacy.

To fill this gap, this study investigated the essential components of effective automated formative feedback through a controlled experiment. The results of this study provide empirical evidence on the efficacy of automated formative feedback of different configurations, and contribute to the understanding of how CS students utilize it for just-in-time learning.

II. Related Works

A. Automated Grading and Feedback

Programming assignments are difficult to assess and provide feedback in a timely manner for many reasons, including multiple possible approaches to problem solving, necessity to test against many cases to reach sufficient test coverage, and different individual coding habits and styles [3], [12], [13]. As student enrollment grows, the first challenge to address was the assessment. As a result, automated grading has been investigated extensively.

Studies before 2010 on this topic tended to focus on automated grading system development and testing [9], [13], [14]. Systems developed in this period of time require instructors to provide representative test cases and manually tune feedback to work effectively. Web-CAT and Autolab are two representative examples [15], [16]. More importantly, early systems and studies had great concerns over the possibility that students may game the system, so such systems typically expected the submission of a fully completed program before providing feedback, limit the number of submissions, and limit the completeness of the feedback [8], [9], [13].

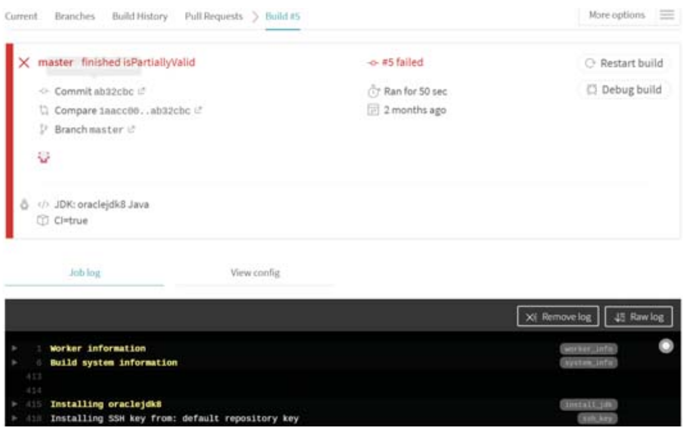
A focus shift from automated grading to automated feedback was witnessed in the most recent decade. Specifically, the focus was on feedback generation through data-driven approaches [17], [18]. Massive Open Online Programming Courses provided large datasets of programming assignments, which is critical to make such approaches possible. The efforts

978-1-7281-0810-0/19/$31.00 ©2019 IEEE

typically aim at providing student suggestions on repairing their program through measuring the distance between their program and the most similar working program [19]–[21]. Such efforts are still in their early stage for two reasons. First, these approaches were rarely tested in authentic environments. Second, how these approaches can be effectively applied to a significantly smaller dataset (i.e., a face-to-face CS1 in a large university) is still unknown.

B. Effective Feedback from Educational Perspectives

In general, feedback can be categorized into two types: formative and summative. Summative feedback is provided when assessment results are released, whereas formative feedback is provided during the learning process. Formative feedback has been found constantly more effective than summative feedback in helping students learn in educational studies across different disciplines, because summative feedback serves more as a justification of the assessment results in student eyes [5], [6], [8]. However, this important perspective was not well taken by studies on automated feedback. Early studies found that students tended to abuse multi-leveled hints where the bottom-level hint revealed direct answers [22]–[24]. Although hints can only be considered as one type of feedback, many studies used the two terms interchangeably [8], [13], [22], [25]. This may contribute to the lack of investigations on how CS students actually use automated feedback.



```
Worker information
Build system information


Installing oraclejdk8
Installing SSH key from: default repository key
```

Fig. 1. A partial screenshot of what a student sees on Travis-CI

III. Research Design

A. Research Questions

Two research questions to guide the research design, including:

1) What feedback component is more essential than others to student learning?
2) How do automated feedback configurations affect student usage behavior?

B. System Implementation and Feedback Design

The automated feedback service we implemented for this study was through the existing infrastructures, including GitHub, GitHub Classroom, Travis-CI and Gradle. Travis-CI is a distributed continuous integration service for building and testing software projects hosted at GitHub [26]. Gradle is an open-source build automation system for Java programming [27]. Our settings allow students to commit & push to GitHub as many times as they want before due dates. Every commit & push will trigger the execution of the submitted code and prepared testing code, and the generated formative feedback will be presented on Travis-CI (https://travis-ci.com; see Figure 1). Students can always get formative feedback regardless of if they have fully finished the program or not. In addition, no explicit submission actions are required beyond regular commits & pushes.

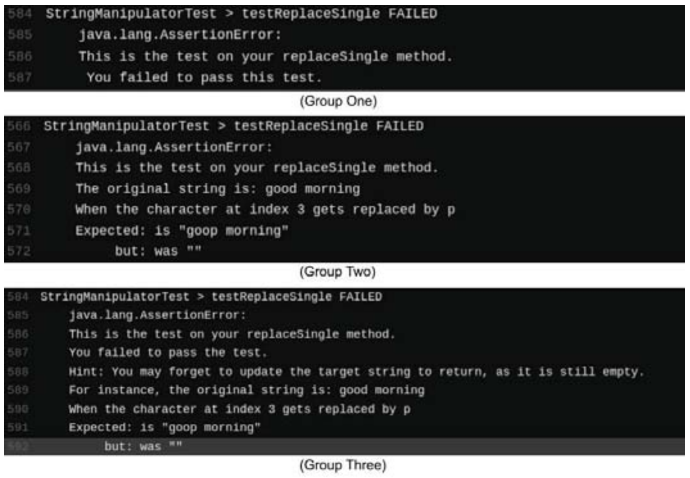
The key of this study is the design of feedback. Synthesizing the literature on feedback, we classified common feedback into three types in the context of computing education:

1) *What's wrong*: Per test case, what it is testing and whether it is a pass or failure [28]
2) *Gap*: Per test case, what the expected output is, and what the actual output is [29]
3) *Hint*: Per test case, how you may fix the issue if the test case fails [22], [25]

All types of feedback were implemented per test case. To effectively implement *Hint*, we adopted the approach used by Parihar et al. [20]. We collected student submissions of the same programming assignments over the last three quarters, summarized the common mistakes and problems, and designed adaptive hints for the top five common mistakes per test case.

C. Experiment Design

This study was conducted in a large university in the North American Pacific Northwest. 46 students taking a CS2 participated in this study. The course was composed of both lectures and lab sessions. Students were expected to complete



```
StringManipulatorTest > testReplaceSingle FAILED
    java.lang.AssertionError:
    This is the test on your replaceSingle method.
     You failed to pass this test.
```

(Group One)

```
StringManipulatorTest > testReplaceSingle FAILED
    java.lang.AssertionError:
    This is the test on your replaceSingle method.
    The original string is: good morning
    When the character at index 3 gets replaced by p
    Expected: is "goop morning"
         but: was ""
```

(Group Two)

```
StringManipulatorTest > testReplaceSingle FAILED
    java.lang.AssertionError:
    This is the test on your replaceSingle method.
    You failed to pass the test.
    Hint: You may forget to update the target string to return, as it is still empty.
    For instance, the original string is: good morning
    When the character at index 3 gets replaced by p
    Expected: is "goop morning"
         but: was ""
```

(Group Three)

Fig. 2. An example of different feedback configurations for a method

three complex individual programming assignments during the lab sessions. Each assignment required about 200-300 lines of code to completely solve the given problem. Students were randomly and evenly divided into three different lab sessions. Each student only attended one lab session throughout the whole semester. Therefore, each lab session was treated as a group. The automated feedback each group received on their programming assignments were configured different (see Figure 2):

1) Group One: *What's wrong*
2) Group Two: *What's wrong + Gap*
3) Group Three: *What's wrong + Gap + Hint*

All students learned how to use Git before taking the experimental course, and they were given detailed instruction on how to utilize automated formative feedback in the beginning of the course. All student coding behaviors captured by Git and Travis were tracked. Students were asked to indicate the frequency of checking feedback on Travis-CI by the end of the course. Their performance on programming assignments were also recorded.

IV. Results

A. What feedback component is more essential than others to student learning?

To answer the question "*What feedback component is more essential than others to student learning?*", we examined and compared student programming assignment performance across the three groups. The performance of 46 students from three groups was summarized in Table 1.

TABLE I
Student average performance per group per programming assignment

Group	Student numbers	Average Performance		
		Assignment 1	Assignment 2	Assignment 3
One	16	70	64.49	79.53
Two	15	95.41	86.25	83.75
Three	15	95	94.17	91.67

Full score of each assignment is 100.

One-way multivariate analysis of variance (MANOVA) was applied to examine the differences in student performance across the three groups. Using Pillai's trace, there was a significant effect being detected, V = 0.655, F(6, 84) = 6.823, $p < 0.01$. The observed statistical power was 0.98.

The MANOVA was followed up with discriminant analysis, which revealed two discriminant functions. The first function explained 98.2% of the variance, cononical $R^2 = 0.63$, whereas the second explained 1.8%, canonical $R^2 = 0.03$. In combination, these discriminative functions significantly differentiated Group One from Group Two and Three, $\lambda = 0.363$, $\chi^2(6) = 42.513$, $p < 0.01$, but removing the first function indicated that the second function did not significantly differentiate the remaining two groups, $\lambda = 0.970$, $\chi^2(2) = 1.280$, $p > 0.05$. In other words, the significant differences detected by MANOVA only existed between Group One and Group Two / Three. No significant difference was found between Group Two and Three.

The findings show that when students only received *What's wrong* feedback, their performance significantly lagged behind their counterparts receiving *Gap* feedback. However, no significant difference was observed between the groups with and without *Hint* feedback.

B. How do automated feedback configurations affect student usage behavior?

To answer the question "*How do automated feedback configurations affect student usage behavior?*", we conducted a one-question survey at the end of the course to (1) confirm that students indeed used the provided automated feedback, and (2) to learn about group differences in terms of feedback usage. The one question in the survey was:

How often did you check the feedback on Travis-CI?

A 4-point Likert scale was adopted for the question. Choices for the question include:

(a) Rarely
(b) Sometimes
(c) Often
(d) Always

The four choices corresponded to points ranging from 1 to 4. The average of all 46 students was 3.72. One-way Analysis of variance showed no significant difference across the three groups, F(2, 43) = 0.124, $p > 0.05$. In other words, students reported that they used the automated formative feedback frequently regardless of which group they were in.

To further understand the group difference in feedback usage, we aggregated student feedback seeking behaviors (i.e., commit & push) by day. When the aggregated commits & pushes are plotted against the time, there is a clear difference in commit & push numbers among the three groups. For instance, students had 15 days to work on programming assignment three (see Figure 3). In the first five days no clear pattern can be found. In the remaining ten days, Group One committed & pushed significantly more frequently than Group Two and Three, especially during the last three days ahead of the due date. However, there is no apparent frequency difference between Group Two and Three. Similar effects were observed on all three programming assignments. Overall, students who only received *What's wrong* feedback committed & pushed more frequently than their counterparts receiving *Gap* feedback. No significant difference was observed between the groups with and without *Hint* feedback.

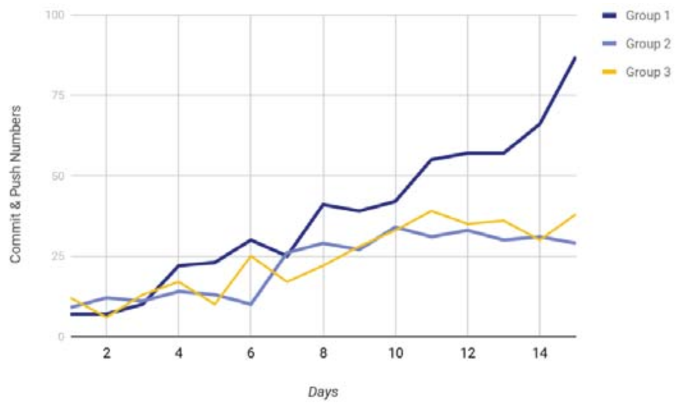


Fig. 3. An example of commit & push numbers per group for a programming assignment

V. Discussion

Among the findings of this study, we would like to highlight two points. First, the observed ”system gaming behaviors” among students might be an indicator of ineffective feedback design instead of students abusing automated feedback intentionally. Given that nearly all students reported substantial usage of automated formative feedback, it is safe to assume that when a student committed & pushed their code to GitHub, they intended to seek feedback. Based on this assumption, it is obvious that students who only received *What's wrong* feedback sought feedback much more frequently than their counterparts in Group Two and Three. If we only look at the commit & push numbers of this group of students exclusively, it is tempting to conclude that this group of students gamed the system and abused automated feedback. Many prior studies that had similar observations interpreted it as ”intentional system gaming behaviors”, and further proposed limiting the number of feedback students can get, or providing no feedback prior to assessment at all [23], [30], [31]. However, when we classified feedback into different types and tested them individually in a controlled study, evidence against this interpretation emerged. Students who only received *What's wrong* feedback sought feedback more frequently, but they did not perform as well as their counterparts receiving more fine-grained feedback. In other words, the reason this group of students sought more feedback is not likely because they were taking advantage of the unlimited feedback. On the other hand, those students might not get the help they needed in problem solving. They kept seeking more feedback simply to use it as a confirmation to see if they successfully passed all test cases, but they were rarely sure if they were on the right track.

Second, the effectiveness of hints delivered by automated feedback deserves further investigation. Another important finding from the results is that there was no significant difference between students who received and those who did not receive *Hint* feedback in either academic performance or feedback-seeking frequencies. There are different ways to interpret this finding. One possibility is that the *Hint* as a feedback type was not implemented well enough to realize its full potential in this study. Only adaptive hints for top five errors per test case were implemented. For errors outside of this scope no adaptive hints were provided. It is possible that the summarized top five errors were not representative enough to cover the errors students made during this experimental course. As the result, students found hints of little help. Some researchers may argue that the lack of multi-level hints is another reason. We intentionally decided not to implement hints in multiple levels in which the bottom level is closest to revealing direct answers. This design was found to stimulate system gaming behaviors and to be detrimental to student learning in studies on intelligent tutor systems [22], [23]. Another possible interpretation is that the comparisons between expected and actual outputs provided sufficient information for students to move forward. It is worth noting that students taking the experimental course were provided multiple channels to have their questions answered, including instructor office hours, teaching assistant office hours, and a dedicated online Question & Answer platform where students can ask learning questions to both instructors and their peers. The process of debugging, research and fixing errors might take time, but also provide students a valuable opportunities to learn debugging [32], [33].

VI. Limitations

This study is not without limitations. The sample size of this study is comparatively small. Although the sample size was sufficient for a three-group controlled experiment, it is unknown whether the findings can be generalized to CS courses with significantly larger sizes, or outside of the context of CS2. Future studies may consider replicating this experiment in the context of a larger CS course, especially a CS1. Additionally, no qualitative data on how students actually used automated feedback were collected from the experiment. Either in-depth interviews or observations can provide richer information on student usage of automated feedback and the relationship between usage and automated feedback configuration. Future studies are recommended to utilize both quantitative and qualitative methods to answer the research question ”*How do automated feedback configurations affect student usage behavior?*”.

VII. Conclusions

This study investigated the essential of meaningful automated feedback for programming assignments using a quasi-controlled experiment. The results revealed that simply knowing what fails does not help students sufficiently, and may stimulate system gaming behavior. Hints were not found impactful on student performance or their usage of automated feedback. In contrast, the gap between the current and expected states seem to provide sufficient information for students to move forward and fix errors in the context of a CS course where multiple support venues were available. We discussed the implications of the findings and further provided guidance on effective automated feedback design based on the findings.

REFERENCES

[1] T. Camp, W. R. Adrion, B. Bizot, S. Davidson, M. Hall, S. Hambrusch, E. Walker, and S. Zweben, "Generation cs: The growth of computer science," *ACM Inroads*, vol. 8, no. 2, pp. 44–50, May 2017. [Online]. Available: http://doi.acm.org/10.1145/3084362

[2] L. J. Sax, K. J. Lehman, and C. Zavala, "Examining the enrollment growth: non-cs majors in cs1 courses," in *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education*. ACM, 2017, pp. 513–518.

[3] B. Cheang, A. Kurnia, A. Lim, and W.-C. Oon, "On automated grading of programming assignments in an academic institution," *Computers & Education*, vol. 41, no. 2, pp. 121 – 131, 2003.

[4] R. Singh, S. Gulwani, and A. Solar-Lezama, "Automated feedback generation for introductory programming assignments," in *Proceedings of the 34th ACM SIGPLAN Conference on Programming Language Design and Implementation*, ser. PLDI '13. New York, NY, USA: ACM, 2013, pp. 15–26.

[5] K. T. Brinko, "The practice of giving feedback to improve teaching," *The Journal of Higher Education*, vol. 64, no. 5, pp. 574–593, 1993.

[6] S. Gielen, E. Peeters, F. Dochy, P. Onghena, and K. Struyven, "Improving the effectiveness of peer feedback for learning," *Learning and Instruction*, vol. 20, no. 4, pp. 304 – 315, 2010.

[7] J. Hattie and H. Timperley, "The power of feedback," *Review of Educational Research*, vol. 77, no. 1, pp. 81–112, 2007.

[8] P. M. Chen, "An automated feedback system for computer organization projects," *IEEE Transactions on Education*, vol. 47, no. 2, pp. 232–240, 2004.

[9] P. Ihantola, T. Ahoniemi, V. Karavirta, and O. Seppälä, "Review of recent systems for automatic assessment of programming assignments," in *Proceedings of the 10th Koli calling international conference on computing education research*. ACM, 2010, pp. 86–93.

[10] S. Safei, A. S. Shibghatullah, and B. Mohd Aboobaider, "A perspective of automated programming error feedback approaches in problem solving exercises," *Journal of Theoretical and Applied Information Technology*, vol. 70, no. 1, pp. 121–129, 2014.

[11] G. Akçapınar, "How automated feedback through text mining changes plagiaristic behavior in online assignments," *Computers & Education*, vol. 87, pp. 123–130, 2015.

[12] M. Sztipanovits, K. Qian, and X. Fu, "The automated web application testing (awat) system," in *Proceedings of the 46th Annual Southeast Regional Conference*. ACM, 2008, pp. 88–93.

[13] P. Guerreiro and K. Georgouli, "Combating anonymousness in populous cs1 and cs2 courses," in *Proceedings of the 11th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education*, ser. ITICSE '06, 2006, pp. 8–12.

[14] L. Malmi, V. Karavirta, A. Korhonen, and J. Nikander, "Experiences on automatically assessed algorithm simulation exercises with different resubmission policies," *Journal on Educational Resources in Computing (JERIC)*, vol. 5, no. 3, p. 7, 2005.

[15] S. H. Edwards and M. A. Perez-Quinones, "Web-cat: automatically grading programming assignments," in *ACM SIGCSE Bulletin*, vol. 40, no. 3. ACM, 2008, pp. 328–328.

[16] G. Haldeman, A. Tjang, M. Babeş-Vroman, S. Bartos, J. Shah, D. Yucht, and T. D. Nguyen, "Providing meaningful feedback for autograding of programming assignments," in *Proceedings of the 49th ACM Technical Symposium on Computer Science Education*. ACM, 2018, pp. 278–283.

[17] J. Gao, B. Pang, and S. S. Lumetta, "Automated feedback framework for introductory programming courses," in *Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education*. ACM, 2016, pp. 53–58.

[18] A. Head, E. Glassman, G. Soares, R. Suzuki, L. Figueredo, L. D'Antoni, and B. Hartmann, "Writing reusable code feedback at scale with mixed-initiative program synthesis," in *Proceedings of the Fourth (2017) ACM Conference on Learning@ Scale*. ACM, 2017, pp. 89–98.

[19] S. Gulwani, I. Radiček, and F. Zuleger, "Automated clustering and program repair for introductory programming assignments," in *Proceedings of the 39th ACM SIGPLAN Conference on Programming Language Design and Implementation*. ACM, 2018, pp. 465–480.

[20] S. Parihar, Z. Dadachanji, P. K. Singh, R. Das, A. Karkare, and A. Bhattacharya, "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, 2017, pp. 92–97.

[21] K. Wang, R. Singh, and Z. Su, "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*. ACM, 2018, pp. 481–495.

[22] I. Roll, V. Aleven, B. M. McLaren, and K. R. Koedinger, "Improving students help-seeking skills using metacognitive feedback in an intelligent tutoring system," *Learning and instruction*, vol. 21, no. 2, pp. 267–280, 2011.

[23] R. Baker, J. Walonoski, N. Heffernan, I. Roll, A. Corbett, and K. Koedinger, "Why students engage in gaming the system behavior in interactive learning environments," *Journal of Interactive Learning Research*, vol. 19, no. 2, pp. 185–224, 2008.

[24] V. Aleven, I. Roll, B. M. McLaren, and K. R. Koedinger, "Help helps, but only so much: Research on help seeking with intelligent tutoring systems," *International Journal of Artificial Intelligence in Education*, vol. 26, no. 1, pp. 205–223, 2016.

[25] M. Eagle and T. Barnes, "Evaluation of automatically generated hint feedback," in *Educational Data Mining 2013*, 2013.

[26] Travis-CI, "Wikipedia - travis ci." [Online]. Available: https://bit.ly/2VZ7QM4

[27] Gradle, "Wikipedia - gradle." [Online]. Available: https://bit.ly/2DrspK0

[28] S. Narciss, S. Sosnovsky, L. Schnaubert, E. Andrès, A. Eichelmann, G. Goguadze, and E. Melis, "Exploring feedback and student characteristics relevant for personalizing feedback strategies," *Computers & Education*, vol. 71, pp. 56–76, 2014.

[29] J. Metcalfe and N. Kornell, "Principles of cognitive science in education: The effects of generation, errors, and feedback," *Psychonomic Bulletin & Review*, vol. 14, no. 2, pp. 225–229, 2007.

[30] R. S. Baker, A. T. Corbett, K. R. Koedinger, and A. Z. Wagner, "Off-task behavior in the cognitive tutor classroom: when students game the system," in *Proceedings of the SIGCHI conference on Human factors in computing systems*. ACM, 2004, pp. 383–390.

[31] R. S. d Baker, A. T. Corbett, K. R. Koedinger, and I. Roll, "Generalizing detection of gaming the system across a tutoring curriculum," in *International conference on intelligent tutoring systems*. Springer, 2006, pp. 402–411.

[32] L. Murphy, G. Lewandowski, R. McCauley, B. Simon, L. Thomas, and C. Zander, "Debugging: the good, the bad, and the quirky–a qualitative analysis of novices' strategies," in *ACM SIGCSE Bulletin*, vol. 40, no. 1. ACM, 2008, pp. 163–167.

[33] J. M. Griffin, "Learning by taking apart: deconstructing code by reading, tracing, and debugging," in *Proceedings of the 17th Annual Conference on Information Technology Education*. ACM, 2016, pp. 148–153.