How Automated Feedback is Delivered Matters: Formative Feedback and Knowledge Transfer

Qiang Hao
Computer Science & SMATE
Western Washington University
Bellingham, WA
qiang.hao@wwu.edu

Michail Tsikerdekis
Computer Science
Western Washington University
Bellingham, WA
michael.tsikerdekis@wwu.edu

***Abstract*—This Full Research Paper investigated the correlations of automated formative feedback and student knowledge transfer in an advanced-level network course. A microservice of automated formative feedback was designed and implemented. The automated feedback was set up to address correctness and coding style along a series of milestones in finishing each of the assigned programming projects. 36 students participated in this study. Different from the concerns of prior studies that automated formative feedback may encourage trial & error and harm learning, this study identified a link between the automated formative feedback and student knowledge transfer. The results of this study confirmed positive findings from other fields on formative feedback, and call for more research on formative feedback in computing education.**

***Index Terms*—automated formative feedback, computing education, programming assignments, knowledge transfer**

I. Introduction

Programming courses are typically characterized by heavy code-writing assignments and large student enrollments. Programming assignments are notoriously difficult to assess in a timely and effective manner [1], [2]. Some programs which pass certain test cases may fail others, while some may lack comments and are difficult to read. Therefore, one of the urgent challenges facing computer science (CS) educators was to automate the grading process [3]. As a complete program is being assessed, feedback can be conveniently provided along with the given grade. As a result, automated grading of programming assignments was studied extensively in the past two decades, and feedback provided along the assessment was typically studied as a subprogram of automated assessment [2], [4], [5].

From the perspective of constructivism, feedback can be either formative or summative [6], [7]. Summative feedback refers to the feedback provided along the assessment results to students and it is typically considered final. In other words, a student has no opportunity to correct their grade. In contrast, formative feedback is provided during the learning process instead of when student performance is being assessed [8]. Formative feedback can help students identify their strengths and weaknesses and target areas that need further work. Prior studies in the fields such as biology, physics, and mathematics found formative feedback tends to be utilized by students more frequently than summative feedback [9]–[11]. In addition, formative feedback was found more effective in improving student learning efficacy and academic performance [12]. However, formative feedback was much less studied in the context of computing education.

To fill this gap, this study investigated the effectiveness of automated formative feedback on CS students' knowledge transfer. The results of this study provide empirical evidence on the efficacy of automated formative feedback on CS students in programming courses, and contribute to the understanding of effective learning & teaching approaches in computing education.

II. Related Works

A. *Constructivism and Feedback*

Constructivism, as a learning theory, views learning as a process of constructing knowledge based on prior knowledge and experience. Learning occurs when students engage with concrete examples, process information, and decontextualize heuristics. This indicates that feedback is effective only when learners act on the information to improve subsequent work [13], [14]. As the result, if students are dissatisfied with assessment results, they are less likely to act on the summative feedback, and thereby less likely to learn.

Empirical studies consistently support the above feedback hypothesis deduced from Constructivism. Formative feedback has constantly been found more effective than summative feedback in terms of addressing knowledge gap and improving student performance [15], [16]. On one hand, summative feedback often functions mainly as a justification of the assessment results and thereby limits its developmental effect [17]. Students are aware of the distinctions between summative and formative feedback, and they tend to disregard the grade-justifying feedback [8], [18]. On the other hand, formative feedback can provide benefits on both cognitive and metacognitive levels to students. In addition to informing students of the faulty interpretation or knowledge gap, formative feedback can help students self-regulate their learning and increase their confidence [19], [20]. Although some studies investigated the possibility of making summative feedback effective, most of them acknowledged its difficulties in practice [21].

Despite of the abundant evidence on the benefits of formative feedback [9]–[11], [22], it was rarely studied in the context of computing education. This phenomenon has been observed by both Ala-Mutka [3] and Keuning et al [15] in their survey studies on automated testing and feedback tools. On one hand, many developed feedback tools in computing education tended to put limits on the number of feedback seeking attempts [3]. On the other hand, many developed tools were no longer accessible by the time when their related papers were published. Such papers typically focused on the tool development, but did not touch on the aspect of feedback delivery (summative vs. formative) [3], [15].

B. Automated Feedback

What sets programming courses apart from many other courses is the difficulty of grading assignments. First, evaluating a program requires running it against multiple test cases, and programs that work on certain test cases may fail on others [1]. Second, multiple approaches of problem solving are usually viable for the same problem. It is difficult for graders to resort to one ”model answer” when grading programs using different approaches [1], [23]. Third, even the same approach is used, two programs may still have vastly different representations depending of individual coding habits and styles. Individual students are likely to use different variable names, comment on different sections, and factorize different pieces of code as functions. It is time-consuming to read code with different structures and styles. Considering the typically large enrollment sizes of programming courses, it is extremely time-consuming to assess programming assignments systematically and efficiently. Therefore, how to automate the assessment of programming assignments emerged as one of the oldest problem to tackle in the community of computing education. As a program is being assessed, summative feedback can be conveniently provided along with the given grades. As a result, feedback was initially studied as a subprogram of automated assessment.

Early studies between 2000 and 2010 on this topic tended to focus on system development and error message design [16], [18], [24]. Most systems developed in this period of time require instructors to provide representative test cases and manually tune feedback to work effectively. Additionally, early studies had great concerns over whether students would abuse feedback, so their systems typically (1) expected students to submit a complete version of the program on their first submission, (2) put limits on number of submissions, and (3) put limits on the amount of provided feedback and the completeness of the feedback [16], [17].

The most recent decade witnessed a focus shift to scalibility of feedback as well as feedback generation through data-driven approaches [25], [26]. Taking advantage of the large dataset of programming assignment submissions to Massive Open Online Programming Courses, such efforts typically provide repair suggestions based on clustering correct solutions and matching between faulty submissions and correct solutions [27]–[29]. Although such methods are fully automated and can be scaled up massively, examinations on such methods from pedagogical perspectives are still lacking. In addition, such methods might be less effective for typical college-level programming courses that have significantly fewer students.

Although automated feedback has been studied from different angles, the delivery approaches of automated feedback (summative or formative) are rarely studied. This surprising yet interesting phenomenon is partially due to the claims from early automated assessment studies that formative feedback might encourage trial and error behaviors, lead to system abusing, and harm student learning [17], [24], [30], [31]. Such claims can be traced to an experience report of Chen [17], which reported that students tended to perform trial and error using formative feedback. However, it might be hasty to conclude that students would barely learn from class or would never realize that trial and error rarely helps solving problems after multiple attempts.

It is worth noting that many studies claiming that formative feedback might be harmful were actually studying automated hints instead of automated feedback. Hints address the next step(s) in problem solving, whereas feedback address the mistakes learners have made. Hint functions were initially studied by research on intelligent tutor systems. Such studies were typically situated in K-12 education, within the context of mathematics or physics education [32]–[34]. Hints in the studied tutor systems were typically organized hierarchically by the distance to revealing the final answers: the hint at the surface level is most limited in information, whereas the hint at the bottom level almost reveals the answers. Such systems often allowed students to reach hints at deeper levels by simply clicking buttons. Such a design was found to encourage system gaming behaviors that harm learning [35], [36]. Although hint and feedback were used interchangeably in many computing education studies, the pedagogical differences between them are significant (e.g., Anderson et al [37] combined both feedback and hint in the LISP Tutor, but only used ”feedback” to refer to both the facilitative functions).

III. Research Design

A. Research Questions

The two research questions guiding this study include:

1) How is using automated formative feedback correlated with knowledge transfer of computer science students?
2) How do computer science students react to automated formative feedback based on their past academic performance?

B. System Design and Development

A microservice was developed in this study to serve the purpose of providing automated formative feedback to students. Microservice refers to a paradigm of developing software applications as modular services serving single purposes and communicating with other software through well-defined application programming interfaces [38]. The developed microservice works in conjunction with both the university-wide learning management system (LMS) and homework repository. The LMS is where students can check their grades and find the automated formative feedback on each of their assignments. The homework repository is where students submit their programming assignments. In addition, the homework repository also performs version control over all assignments. The developed microservice communicates with both the LMS and homework repository through their Application Programming Interface (API). Once an assignment is submitted or resubmitted to the homework repository, the microservice is triggered to examine the submitted code and push feedback to the LMS [see Fig. 1].

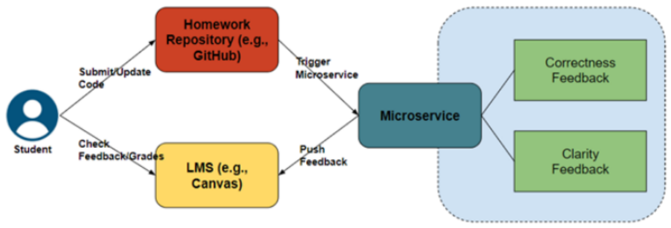

Fig. 1. Architecture of the Microservice

C. Experiments

In this study, the developed microservice was deployed in a network course with 36 computer science majors at a comprehensive public university in the Northwestern United States. The institutionally adopted LMS of the university is Canvas (https://www.instructure.com/), and the homework repository being used for the course was GitHub (https://github.com/). Students completed a series of complex programming-intensive projects in the course. All projects required the application of the learnt conceptual knowledge in network. The feedback seeking behavior of individual students were tracked and collected during the whole course time.

To fully test the efficacy of automated formative feedback, we did not set limits on either the number of feedback seeking attempts or the amount of provided feedback. In addition, our microservice provides feedback whether the program is complete or not by the due date. To realize this, a sequence of milestones were predefined for each project, and feedback by milestone instead of final results were provided [see Fig. 2]. It is worth noting that if students fail to pass a milestone, the testcases on the sequential milestones will not be triggered to provide more feedback. Moreover, passing each milestone will result in earning partial credit. The credits were reflected in individual student accounts of Canvas immediately.

Feedback delivered by our microservice focused on two aspects of programming: correctness and clarity. Correctness feedback was generated through running the submitted code against a series of tests with increasing complexity. For instance, a simple test case examines whether a port was bound by a server, whereas a more complex test case evaluates network concurrency by involving submitted bytes from multiple clients. A failed test will be reported to individual students with readable information and references to learning

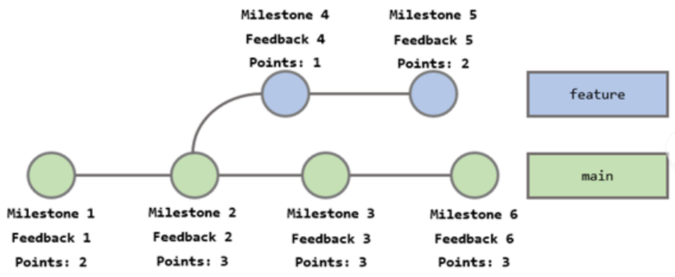

Fig. 2. Sample Milestones of a Project

resources, so students can learn further and address the problem. Correctness feedback was realized through sandboxing and using randomization Unix tools (e.g., bats and timeout). Clarity feedback refers to feedback on students' coding readability and styles. Such feedback specifically addresses issues such as inconsistent naming conventions, code redundancy, or obvious inefficiency. A static checker, *pylint3*, was used to provide clarity feedback to students. Pylint3 checks coding styles and readability by PEP8 Style Guide for Python Code (https://www.python.org/dev/peps/pep-0008).

D. Measurements

Student feedback seeking frequency was measured by counting the number of submission and resubmissions of their programming assignments. As is shown in Fig. 1, when an assignment is submitted to the homework repository, the microservice will be triggered to examine the submitted code and provide feedback accordingly. This measurement is consistent with prior studies on automated programming assessment [17], [18].

By the end of the course, a final exam on Computer Networks was given to all students to measure the knowledge transfer. The exam was a commonly used formal final exam for the course that included topics commonly found in computer networks such as Ethernet Protocol, TCP and UDP protocols, Socket API and packet formatting. For example, simple questions focused on concepts such as what is the broadcasting IP at layer 2 on the OSI model. More advanced questions focused on the functional aspects of programming such as "In the socket API which of the following functions is not used by a server application." The expectation was that students that learn to do well with computer network programming will have a better understanding of the theory and as such perform better in the final exam. The exam was multiple choice and was automatically graded by the e-learning platform (Canvas).

A survey was distributed to collect demographic information and inquire about student preference of the automated formative feedback:

1) What is your gender?
2) What is your age?
3) What college year are you in?
4) Do you like automated formative feedback? (*Four-point Likert scale was used*)

5) Do you find automated formative feedback helpful? (*Four-point Likert scale was used*)

The four-point Likert scale had the following choices: Strongly Agree, Agree, Disagree, Strongly Disagree. Student academic performance in prior CS courses was collected from the institutional data warehouse. In addition, an in-depth interview with randomly selected students was conducted to better understand their reactions to the automated formative feedback.

IV. Results

A. *Descriptive Summary*

A total of 36 CS students enrolled in an advanced-level CS course participated in this study. Their demographic information is summarized in Table 1.

TABLE I
DESCRIPTIVE SUMMARY OF PARTICIPANTS

Number of students	36
Gender	
Male	30.6%
Female	69.4%
Age	
$<= 21$	22.9%
> 21 and $<= 23$	42.8%
> 23	34.3%
College Year	
3rd Year	11.1%
4th Year	88.9%

B. *How is using automated formative feedback correlated with knowledge transfer of computer science students?*

Given that automated formative feedback directly addressed issues in course projects, it was natural if students who sought feedback more frequently tended to have better performance on the programming projects. This has been confirmed by prior studies on automated formative feedback [16], [18]. However, it is unknown whether digesting and absorbing the feedback is correlated with student knowledge transfer from the projects to other aspects of the course.

All students in this study had taken at least five CS courses prior to the current course. Their cumulative CS GPA was collected from the university academic records, given that prior course performance of a student is strongly predictive of his or her current and future performance. Correlational analysis revealed that seeking feedback, cumulative CS GPA, and knowledge transfer are all positively correlated with each other [see Table 2]. All the correlations were found to be significant.

TABLE II
CORRELATIONAL ANALYSIS AMONG SEEKING FEEDBACK, CUMULATIVE CS GPA, AND KNOWLEDGE TRANSFER

	SF	CGPA	KT
Seeking Feedback (SF)	1.0		
Cumulative CS GPA (CGPA)	.422*	1.0	
Knowledge Transfer (KT)	.380*	.627**	1.0

* $p < .05$; ** $p < 0.01$; *** $p < .001$

To explore the relationship between seeking feedback and cumulative CS GPA, we grouped students into two groups: those with below-average CS GPA and those with above-average CS GPA. Students with above-average CS GPA tended to seek feedback more frequently than their counterparts. Students with above-average CS GPA sought feedback 61 time on average. In contrast, their counterparts sought feedback 41 times on average.

As the next step, we further grouped students by their frequency of seeking feedback. For students with above-average CS GPA, no significant difference was found between those who sought feedback more than average times and those who sought feedback less than average times, $t(12) = 0.82, p = 0.43$. In contrast, for students with below-average CS GPA, those who sought feedback more than the average times significantly outperformed their counterparts in the exam, $t(11) = 2.50, p < .05$ [see Table 3].

TABLE III
STANDARDIZED FINAL EXAM PERFORMANCE OF STUDENTS GROUPED BY CUMULATIVE CS GPA AND FREQUENCY OF SEEKING FORMATIVE FEEDBACK

Group Criteria 2: Seeking formative feedback	Group Criteria 1: Cumulative CS GPA	
	Below-Average	Above-Average
Less than average time	-1.06	0.34
More than average time	0.2	0.51

C. *How do Computer Science Students React to Automated Formative Feedback?*

Two survey questions (Q4 ”Do you like automated formative feedback?” and Q5 ”Do you find automated formative feedback helpful?”) [see Appendix A] and in-depth group interviews were used to measure students’ reactions to automated formative feedback. Students’ responses to the survey questions (Q4: mean 3.39; Q5: mean 3.33) indicated that most students enjoyed seeking automated formative feedback and found it helpful in finishing assigned projects.

Four randomly selected students were invited for the interview. Each student was interviewed for 15 to 20 minutes. All interviews were audio-recorded, annotated and transcribed. Thematic analysis was used to guide the analysis of the interview data. Two rounds of coding were conducted on the data. The first round was to establish a consistent coding guideline. The second round was to apply the established guidelines and fix prior errors. Core themes were identified through comparing data across different interviewees. Two key themes emerged from the analysis were time management and motivation.

Students being interviewed expressed two aspects of automated formative feedback — immediacy and unlimited number of testing — helped them know where they stand in terms of finishing the assigned projects, and better manage their time of working on the projects. Working in concert with Canvas and GitHub, the microservice could inform students of the partial credits they earned and the current average of the whole class, which was deemed motivating for them to start early and commit more time to the assignments. One student said that:

> For me it is good to check to see where I am. I feel that I am motivated by doing more. Every time I updated my code, I can check and see either I get some points or where I did wrong. I can also compare myself to the whole class on Canvas, to see if I am behind the class or ahead of, which is something I did a lot. Also it helps me to have confidence in my code as well ......

V. Discussion

Among the findings of this study, we would like to highlight three points. First, this study confirms the findings on the relationship between seeking formative feedback and academic achievements in other fields. Same as many prior studies [9], [11], [22], this study found that seeking formative feedback was positively related to student academic achievement, such as knowledge transfer, confidence and self-regulation from both correlational analysis and the interview with random students. However, different from many prior studies, this study focused on knowledge transfer instead of student assignment performance. Given that formative feedback directly addresses challenges and knowledge gaps in programming projects, it is possible that a positive cause-and-effect relationship exists between seeking formative feedback and student knowledge transfer. The results of this study confirm that the correlation between seeking formative feedback and academic achievements in programming courses is similar as in other fields.

Second, there is a limited (due to sample size) indication that students with weaker prior academic performance benefit more from seeking automated formative feedback. For students with lower-than-average cumulative CS GPA, if they seek more formative feedback, they tend to achieve significantly better knowledge transfer than their counterparts who seek less formative feedback. This difference was not significant for students with higher-than-average cumulative CS GPA. One possible explanation is that formative feedback gives students more opportunities to gauge their misunderstandings and address their knowledge gap, and automation amplifies such an effect [24], [39]. In addition, automated formative feedback might be more suitable for addressing less-complex knowledge components. As such, the effect might be more obvious on students with weaker prior academic performance. Future studies need to seek larger sample size that have a better representation of the groups in order to further investigate this effect.

Third, more facilitation is needed to help students with weaker academic performance to seek automated formative feedback. It is worth noting that students with better-than-average cumulative CS GPA still sought formative feedback more frequently than their counterparts, despite that the feedback seeking frequency did not have significant effect on their knowledge transfer. As is observed in face-to-face contexts, students who need the help the most tend to seek least help [40]. Therefore, even with automated formative feedback deployed, there is still room to raise the awareness of students to use it, especially students with weaker academic performance. Future studies may consider exploring and comparing different approaches to motivate students to use automated formative feedback in programming courses.

VI. Limitations

This study is not without limitations. First, the experiment was conducted in an advanced-level computer science course, where students have taken at least five core computer science courses prior to the current one. The findings on students at this level might not be the same as on entry-level students. Therefore, the extent to which the findings of this study can be generalized needs further exploration. Second, the cause-and-effective relationship between seeking automated formative feedback was not examined. Although the findings of this study confirmed the positive correlations between the two factors, correlation analysis by itself can not fully confirm the benefits of automated formative feedback. Future studies may consider path analysis on the two factors with a biger sample size. In addition, in the experiment whenever student push their committed updates on programming projects to GitHub, automated formative feedback would be triggered. We assumed that students would not make such pushes for no apparent reasons except for seeking automated formative feedback. Such assumption may not stand equally true for entry-level courses where students are less familiar with basic commands of Git. Future studies are recommended to adapt the assignment submission schemes when focusing on entry-level programming courses.

VII. Conclusions

Over the past few years there have been numerous studies showing the benefits of formative feedback in different fields. However, there has been a limited interest in the delivery of feedback except for its automation and scalability in computing education. This study highlighted the benefits of automated formative feedback on CS student knowledge transfer in a network course. Building on the findings of this study, we call for more empirical assessment on automated feedback in computing education with an emphasis on different delivery approaches.

References

[1] 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.

[2] 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.

[3] K. M. Ala-Mutka, "A survey of automated assessment approaches for programming assignments," *Computer Science Education*, vol. 15, no. 2, pp. 83–102, 2005.
[4] S. H. Edwards, "Rethinking computer science education from a test-first perspective," in *Companion of the 18th Annual ACM SIGPLAN Conference on Object-oriented Programming, Systems, Languages, and Applications*, ser. OOPSLA '03. New York, NY, USA: ACM, 2003, pp. 148–155.
[5] R. Pettit, J. Homer, R. Gee, S. Mengel, and A. Starbuck, "An empirical study of iterative improvement in programming assignments," in *Proceedings of the 46th ACM Technical Symposium on Computer Science Education*, ser. SIGCSE '15. New York, NY, USA: ACM, 2015, pp. 410–415.
[6] K. T. Brinko, "The practice of giving feedback to improve teaching," *The Journal of Higher Education*, vol. 64, no. 5, pp. 574–593, 1993.
[7] 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.
[8] J. Hattie and H. Timperley, "The power of feedback," *Review of Educational Research*, vol. 77, no. 1, pp. 81–112, 2007.
[9] S. M. Paul Orsmond and K. Reiling, "Biology students utilization of tutors' formative feedback: a qualitative interview study," *Assessment & Evaluation in Higher Education*, vol. 30, no. 4, pp. 369–386, 2005.
[10] J. Fluckiger, Y. T. y Vigil, R. Pasco, and K. Danielson, "Formative feedback: Involving students as partners in assessment to enhance learning," *College Teaching*, vol. 58, no. 4, pp. 136–140, 2010.
[11] K. Ludvigsen, R. Krumsvik, and B. Furnes, "Creating formative feedback spaces in large lectures," *Computers & Education*, vol. 88, pp. 48 – 63, 2015.
[12] J. Gikandi, D. Morrow, and N. Davis, "Online formative assessment in higher education: A review of the literature," *Computers & Education*, vol. 57, no. 4, pp. 2333 – 2351, 2011.
[13] D. Wiliam, "What is assessment for learning?" *Studies in Educational Evaluation*, vol. 37, no. 1, pp. 3 – 14, 2011.
[14] B. O'Donovan, C. Rust, and M. Price, "A scholarly approach to solving the feedback dilemma in practice," *Assessment & Evaluation in Higher Education*, vol. 41, no. 6, pp. 938–949, 2016.
[15] H. Keuning, J. Jeuring, and B. Heeren, "Towards a systematic review of automated feedback generation for programming exercises," in *Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education*, ser. ITiCSE '16, 2016, pp. 41–46.
[16] 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.
[17] P. M. Chen, "An automated feedback system for computer organization projects," *IEEE Transactions on Education*, vol. 47, no. 2, pp. 232–240, 2004.
[18] 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.
[19] R. Moreno and A. Valdez, "Cognitive load and learning effects of having students organize pictures and words in multimedia environments: The role of student interactivity and feedback," *Educational Technology Research and Development*, vol. 53, no. 3, pp. 35–45, 2005.
[20] V. J. Shute, "Focus on formative feedback," *Review of educational research*, vol. 78, no. 1, pp. 153–189, 2008.
[21] C. Evans, "Making sense of assessment feedback in higher education," *Review of educational research*, vol. 83, no. 1, pp. 70–120, 2013.
[22] J. M. Van De Ridder, K. M. Stokking, W. C. McGaghie, and O. T. J. Ten Cate, "What is feedback in clinical education?" *Medical education*, vol. 42, no. 2, pp. 189–197, 2008.
[23] 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.
[24] 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.
[25] 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.
[26] 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.
[27] 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.
[28] 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.
[29] 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.
[30] 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.
[31] G. Akçapınar, "How automated feedback through text mining changes plagiaristic behavior in online assignments," *Computers & Education*, vol. 87, pp. 123–130, 2015.
[32] J. A. Walonoski and N. T. Heffernan, "Detection and analysis of off-task gaming behavior in intelligent tutoring systems," in *International Conference on Intelligent Tutoring Systems*. Springer, 2006, pp. 382–391.
[33] 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.
[34] M. Cocea, A. Hershkovitz, and R. S. Baker, "The impact of off-task and gaming behaviors on learning: immediate or aggregate?" in *Proceeding of the 2009 conference on Artificial Intelligence in Education: Building Learning Systems that Care: From Knowledge Representation to Affective Modelling*. IOS Press, 2009.
[35] R. S. Baker, A. de Carvalho, J. Raspat, V. Aleven, A. T. Corbett, and K. R. Koedinger, "Educational software features that encourage and discourage gaming the system," in *Proceedings of the 14th international conference on artificial intelligence in education*, 2009, pp. 475–482.
[36] R. S. Baker, "Stupid tutoring systems, intelligent humans," *International Journal of Artificial Intelligence in Education*, vol. 26, no. 2, pp. 600–614, 2016.
[37] J. R. Anderson, F. G. Conrad, and A. T. Corbett, "Skill acquisition and the lisp tutor," *Cognitive Science*, vol. 13, no. 4, pp. 467–505, 1989.
[38] M. Villamizar, O. Garcés, H. Castro, M. Verano, L. Salamanca, R. Casallas, and S. Gil, "Evaluating the monolithic and the microservice architecture pattern to deploy web applications in the cloud," in *Computing Colombian Conference (10CCC), 2015 10th*. IEEE, 2015, pp. 583–590.
[39] M. Taras, "Assessment–summative and formative–some theoretical reflections," *British journal of educational studies*, vol. 53, no. 4, pp. 466–478, 2005.
[40] Q. Hao, E. Wright, B. Barnes, and R. M. Branch, "What are the most important predictors of computer science students' online help-seeking behaviors?" *Computers in Human Behavior*, vol. 62, pp. 467–474, 2016.

A. APPENDIX

Survey questions:

1) What is your gender?
2) What is your age?
3) What college year are you in?
4) Do you like automated formative feedback?
5) Do you find automated formative feedback helpful?

Four-point Likert Scale was used for question 4 and 5.