Edukasiana: Jurnal Inovasi Pendidikan
ISSN 2809-6770
Volume 3, Issue 2, 2024, pp. 257-268

Engaging Students to Learn Coding in the AI Era with Emphasis on the Process

Kate Arendes[1], Shea Kerkhoff[2], Badri Adhikari[1]*
[1]Department of Computer Science, University of Missouri – St. Louis, St. Louis, United States
[2]Department of Educator Preparation and Leadership, University of Missouri – St. Louis, St. Louis, United States
*Corresponding Author: adhikarib@umsl.edu

Article History:
Received 2024-02-28
Accepted 2024-05-14

DOI:
10.56916/ejip.v3i2.728

ABSTRACT
Students learning to code for the first time face several challenges. For instance, they struggle to interpret the error messages they see when their code fails to run. Since teaching standards in coding are focused primarily on whether a student's code runs successfully, students are often penalized in their grades not for the effort they put into their work but for the code they turn in. Automated grading tools deployed in educational institutions, unfortunately, make the issue worse. In this work, we discuss a novel approach to motivate students to learn coding by shifting the focus of both educators and students from outcomes to the process behind the outcomes. In this study, educators and students are introduced to a new tool, Process Feedback (PF), which shows each student's work as a visual journey. After using both qualitative and quantitative data collection and analysis, the paper discusses how using PF can help students code insightfully and help educators grade quickly and thoroughly. Our findings reveal that tools such as PF can be beneficial in addressing challenges in learning to code and encouraging students to be original in the age of AI. The results also imply that the incorporation of process-oriented learning tools can make coding education more effective and that the process-centric approach can play a key role in the development and effectiveness of educational tools for students.
Keywords: *process; learning; programming; artificial intelligence; education*

1. INTRODUCTION

Regardless of whether computer programming learners use AI as a copilot, the skills to critically analyze a piece of code are necessary for most successful careers in computing. Unfortunately, learning to code is challenging for beginners. Many of these challenges stem from the fact that the audience of a code is a computer, rather than a human. In many cases, even a minor misspelling in a line of code will hinder the compilation of an entire code/program (Hayes & Mouradian, 1981). To make matters worse, the immediate feedback that the computer "reader" provides comes in the form of complex error messages that assume a level of expertise not possessed by novice and intermediate programmers (Nienaltowski, Pedroni, & Meyer, 2008). Although these error messages typically include an explanation of the error and its location in the code (Marceau, Fisler, & Krishnamurthi, 2011), many learners find them to be "cryptic and hard to understand" (Prather, Pettit, McMurry, Peters, Homer, Simone, & Cohen, 2017) (Hristova, Misra, Rutter, & Mercuri, 2003). Instead of building self-efficacy, these errors can make learners feel helpless.

For instructors too, grading student code is a challenge for at least two reasons. Firstly, a given programming question can be solved in a multitude of ways. Second, a plethora of code fragments is

available online and students often use them instead of learning how to code themselves (Yadav, Burkhart, Moix, Snow, Bandaru, & Clayborn, 2015). As a solution, many educators have turned to two categories of automated tools to make their evaluation of correctness and originality more feasible: plagiarism detection and auto-grading. To identify the submission of unoriginal work, several automatic plagiarism detection tools that evaluate the similarity of code segments have been developed (Devore-McDonald & Berger, 2020). Despite the popularity of these tools, students can still evade detection, as 'similarity' is not a definitive determinant of plagiarism (Kustanto & Liem, 2009). Moreover, plagiarism detection software does not detect code generated by AI. Similarly, automatic grading tools, which have been in use for several decades have limitations (Mekterović, Brkić, & Horvat, 2023) (Ullah, Lajis, Jamjoom, Altalhi, Al-Ghamdi, & Saleem, 2018). If a student's code is mostly correct but does not produce precisely the expected output, students are heavily penalized and their effort is not acknowledged (Ullah *et. al.*, 2018). Similarly, if the automatic system expects a student to code following a particular recipe and the student proposes something new to achieve the same outcome, the student may also be penalized (Ullah *et. al.*, 2018). In summary, plagiarism detection and auto-grading are both ineffective.

These challenges in teaching and learning coding revolve around one theme: not focusing on the process behind students' work. Plenty of evidence in educational research shows that the process of learning and creating is at least as important as the outcome (Shaffer & Resnick, 1999) (Graham & Perin, 2007). For instance, in the process approach to teaching writing, teachers show students that writing is a recursive process involving phases of revision, peer critique, and editing, and that even final drafts are always open to revision (Pritchard & Honeycutt). Without exposure to the complexity of the writing process, students are left with unreasonable expectations, such as the belief that professional authors compose a single draft that is published with only minor revisions (Pritchard & Honeycutt).

Many of the common frameworks for teaching writing instruction, such as the cognitive/motivational and sociocultural models, share characteristics with the process approach, including planning and writing as a nonlinear process, respectively (Cutler & Graham, 2008) (Hayes, 1996) (Schultz & Fecho, 2000). The skills-based approach, which focuses on the "systematic instruction of basic writing skills," on the other hand, distinguishes itself from the process approach in that emphasis is placed on improving specific writing competencies such as spelling and sentence construction (Cutler & Graham, 2008) (Graham & Sandmel, 2011). While it has been shown that skills-based strategies such as supplemental spelling instruction can improve students' sentence construction, this approach does not improve the quality of students' writing. Students who receive process-oriented writing instruction can be better writers than those that receive a traditional or skills-based instruction (Graham, Harris, & Chorzempa, 2002) (Graham & Sandmel, 2011).

Unlike traditional outcome-centric approaches, focusing on students' processes and encouraging students to reflect on them can also develop higher-order thinking skills such as planning, evaluating, monitoring, and reflecting (Livingston, 2003). As with teaching the process for prose writing, emphasizing the process for writing code can assist learners in producing programs that execute properly. Teaching methods and technologies that focus on the process can be powerful alternatives to the existing outcome-centric approaches. An example of such a teaching method is a live coding demonstration where a teacher or a peer student codes 'live' in front of other students. One example of a process-focused tool is Process Feedback (Adhikari, 2023), an online platform that visualizes a learner's coding process for students to self-reflect and for educators to provide feedback on the student's process.

In this article, the authors examine the effectiveness of process-oriented tools and techniques in teaching and learning college-level computer programming. Using a combination of qualitative and quantitative methods, we explore educators' and students' perspectives on process-focused methods, such as live coding and process-revealing coding. The methods section describes the tools used by the students and the approach to collecting data from both educators and students. The results section presents the findings from the analysis of survey and interview data.

2. METHODS

To study the effectiveness of process-oriented tools in teaching and learning computer programming, we conducted case studies in a Department of Computer Science at a large, urban research university in the United States. In the first case study, fourteen students in a Python programming class were exposed to 'live' coding by a Graduate Teaching Assistant highlighting the process. This was followed by a survey that the students completed about their individual assessment of the live coding activity. In the second case study, three instructors and four students who were actively using process-revealing tools for their assignments and grading were interviewed. While the survey yielded quantitative results, the interviews provided us with qualitative insights (Rowley, 2014). The motivation for utilizing interviewing was to ensure that respondents understood the meaning of the questions being posed and provide in-depth data. The response accuracy of flexible interviewing, where an interviewer can alter the phrasing of a question to clarify its meaning to respondents, is significantly greater than standardized survey interviewing when respondents are unsure of the implications of their responses (Schober & Conrad, 1997). Through flexible interviewing, respondents could bring up issues with the tool rather than be limited to a predetermined set of questions. However, both approaches helped evaluate the impact of the tool. For the case studies, the online coding platform Process Feedback (Adhikari, 2023), was utilized. It can be accessed at www.processfeedback.org. In the subsequent sections, we introduce and describe the features of the Process Feedback tool, our live coding demonstration and survey questions asked, and our interview process with the educators and students.

The Process Feedback Tool

Although the issues persisting in programming education could be explored with a large number of tools, this research used Process Feedback (Adhikari, 2023), a free and highly accessible online platform for both writing and coding. The following case studies provide an opportunity to focus on the tool's coding features. Process Feedback was created to aid learners and instructors by visually representing how a piece of code evolves over time. As a user types their code, the web application takes snapshots of their code, and at any point in the process, a report can be generated showing data visualizations summarizing their process at any stage. A version of the user's code is saved every five seconds, allowing the user to generate a process report that contains visualizations displaying data about every version of the content. Figure 1 and Figure 2 are two examples of visualizations that users see. These data have the potential to reveal insights into students' coding approaches, thereby enhancing students' metacognitive skills and reducing instructors' assessment times. The process report generated by Process Feedback contains the following process visualizations:

1. Active Typing and Thinking Durations. The time intervals when a student was typing or thinking or being idle over the duration of coding (see Figure 1).
2. Process Playback. An animation that plays the contents of the text editor throughout the time points.

3. Revision Version Comparison. A split editor with side-by-side comparison of two versions of the code, each of which can be set to any time point.
4. Amount of Text Change Over Time. A timeline with bubbles representing additions and deletions of text shown in green and red color.
5. Code Execution History. A timeline with points representing when the code was executed; Red bubbles indicate failed executions and green bubbles indicate successful executions (see Figure 2).
6. Total Characters and Typing Speed. A chart that combines a line representing the total number of characters with points representing the typing speed at each time point.
7. Active Lines and Line Evolution. A stacked bar chart that shows how each block/line of code has evolved over all time points and highlights lines that were added or edited at each time point.

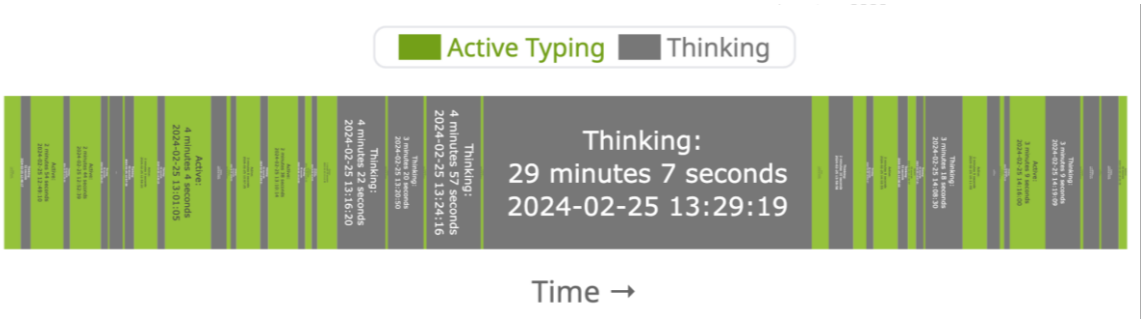

Figure 1. Example of an Active Typing and Thinking bar chart. Active typing durations while working on the task shown as green bars. In the interactive version of the chart, users can select a region to zoom in/out.

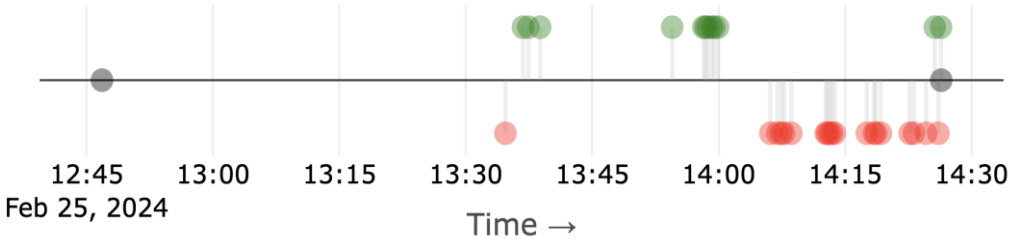

Figure 2. Example of a code execution history chart. Green and red dots correspond to successful and failed code executions. In the interactive version of the chart, hovering over the dots shows the actual output or error message observed.

Case Study I. Live Coding Demonstration

Live coding is the process of designing and implementing a coding project during a lecture (Grønli & Fagernes, 2020). Some examples of live coding include writing a program from scratch, contributing new lines to prewritten code, or copying and pasting prewritten code into the outline of a larger program (Grønli & Fagernes, 2020). While there is an ongoing debate as to whether live coding is always a more effective strategy than having students study static (prewritten) code (Selvaraj et al., 2021), a study has shown that the use of live coding improves students' performance in an introductory computer science course (Rubin, 2013). The effectiveness of live coding stems from its step-by-step demonstration of the coding process as well as providing students with a completed program to refer to when coding independently (Kölling & Barnes, 2004). However, the positive impact of live coding in programming instruction is theoretically limited. While many studies on live coding use the cognitive apprenticeship (CA) theoretical framework as a basis for supporting its use in teaching programming, live coding only fulfills the modeling stage of the CA framework (Selvaraj *et.*

al., 2021). Subsequent stages in the framework include coaching, reflection, articulation, and exploration (Dennen & Burner, 2008). When an instructor completes a program through the use of live coding, the professor is not required to thoroughly explain their cognitive process, such as deciding how often to execute the code and how to interpret error messages. To better leverage the CA framework and implement metacognitive strategies, live coding must be supplemented with means of reflection and articulation.

To determine the impact of a process-oriented live coding demonstration, fourteen students in a "Python for Scientific Computing" course observed a graduate teaching assistant (GTA) solve a coding question provided by the instructor. While coding and utilizing Process Feedback, the GTA described their thought processes. Once the GTA finished solving the problem, the teaching assistant displayed the process report, described the process visualizations to students, and reflected on the visualizations. After watching the presentation of the coding problem, students completed a questionnaire that quantitatively assessed how the tool impacted their learning and provided qualitative feedback on the tool. Students ranked the five following statements from 1 (Strongly Disagree) to 5 (Strongly Agree):

1. Seeing the instructor model their programming process helped me learn about programming.
2. Hearing the instructor think aloud while programming helped me learn about programming.
3. Seeing the visualizations (charts and images) in the digital tool helped me learn how to program.
4. Hearing my professor think aloud to reflect on the data in the visualizations (charts and images) helped me learn about programming.
5. The digital tool improved my learning.

Students also answered two open-ended questions:

1. What was the most helpful part of the Process Feedback tool?
2. What could be improved in the Process Feedback tool?

Case Study II. Instructor Interviews

Table 1. This table lists three instructors who were interviewed in this study along with their course programming language, the number of students in the class, and the duration of the interview with the instructor.

Instructor	Course	Student Count	Interview Duration
#1	Python	23	48 minutes
#2	Java	21	22 minutes
#3	C++	31	33 minutes
Total		75	103 minutes

At a university in the United States, three courses teaching different programming languages were selected as sites for interviewing instructors and students: a Python course, a Java programming course, and an introductory C++ programming course. Each course had its dedicated coding interface where the students could see their course name on the top left corner of the Process Feedback tool. In each course, all students completed all of their coding assignments using Process Feedback and submitted their process report to their instructors once they completed their work. Instructors then examined the students' code and process reports, and then graded the students' work in the institution's learning management system as usual. For instance, one of the assignments in the C++ programming on the topic of 'functions' was for students to write a code for playing the 'Rock, Paper, Scissors' game using functions. Towards the end of the term, the instructors were interviewed via Zoom, asking them several quantitative and qualitative questions. The purpose of the questions was

to determine how Process Feedback impacted teaching and learning in each course. Some of the questions were:

1. Did using the tool allow you to better understand the thought process of your students as they worked on their programs? If so, can you give an example of an instance in which you were able to better understand a student's process?
2. Did using the tool give you any insight into why a student succeeded or struggled in their work? Did using the tool give you a better understanding of how much your students were learning in your course?
3. Was the tool easy to incorporate into your teaching plan? Are there ways in which this tool could have been better incorporated into your teaching plan?
4. If a student asked for feedback on their work, would having their process report available allow you to provide better feedback?
5. Did you identify any improvement in student performance after they used the tool?
6. Overall, how do you think the tool impacted your classroom environment, teaching, and students' learning?

These interview questions were written by the first author based on the overarching research question to elicit perceptions and experiences of the research participants in greater depth. Interviews were audio-recorded and transcribed verbatim by the first author, who then conducted in the first round of content analysis. All three authors reviewed the data, codes, and themes, and reached a consensus that the resulting themes described the data comprehensively. Upon completion, recordings of each interview were summarized into an analytic memo by the first author. The second author used the memos from all the interviews for futher analysis, and the all authors reviewed this analysis to reach an agreement.

Case Study III. Student Interviews

Table 2. This table lists four students who were interviewed in this study along with the degree they were pursuing, the language they were coding in, and the duration of the interview with them.

Student	Pursuing Degree	Coding Language	Interview Duration
#1	CS Graduate	C++	30 minutes
#2	CS Undergraduate	C++	17 minutes
#3	CS Undergraduate	C++	23 minutes
#4	CS Graduate	Python	15 minutes
Total			85 minutes

In all three instructors' courses, students were also required to self-explore their coding process. This enabled us to collect qualitative and quantitative data on how students perceived the use of Process Feedback. In addition these students, we also were interested in studying how students who were not required by their instructors to utilize the tool perceived the use of Process Feedback. We interviewed via Zoom two students from the three courses where Process Feedback was required and two additional students who used the tool but were not a part of the three courses. Students' perception of the use of Process Feedback was evaluated by interviewing students themselves. During the interviews, students discussed how the tool impacted their programming process, including problem-solving, writing lines of code, and debugging. Examples of the questions posed to identify how the use of Process Feedback impacted the students' learning included:

1. How comfortable did you feel showing your professor your programming process?
2. Do you feel that showing your process report to your professor will help your professor provide you with better feedback?

3. Do you feel that using the tool helped you complete your assignment in a more effective manner?
4. How do you think the tool impacted the classroom environment, your professor's teaching, and your learning?

Similar to the instructors' interviews, the student interview recordings were summarized into analytical memos by the first author, which the second author then used for analysis. This analysis was subsequently reviewed and agreed upon by all three authors.

Data Analysis

In the data analysis phase, both quantitative and qualitative data were meticulously examined to ensure robustness and accuracy in analysis and findings. The survey data collected using Google Forms were translated to a comma separated values (CSV) file and imported using a Python script to obtain descriptive statistics and visualized using the Matplotlib library. For each survey question, the responses were reviewed and percentage of responses belonging to each of the response categories were calculated. All data and results were reviewed to ensure validity. Similarly, qualitative data from open-ended survey questions and interviews were processed using thematic content analysis. This involved initial coding followed by a more focused analysis to identify recurring themes and patterns that elucidate the impact of the Process Feedback tool on student learning and instructor teaching methodologies. This dual approach allowed us to comprehensively interpret the effectiveness of Process Feedback in enhancing programming education.

3. RESULTS AND DISCUSSION

In this section, we present and discuss the results from the case studies, which include student survey responses following the live coding demonstration, as well as insights gained from interviewing both instructors and students. The results are organized into major themes that reflect the previously mentioned challenges in programming education, as well as new topics that emerged from the surveys and interviews.

Students Find Live Coding Helpful

Students generally agreed that using Process Feedback during the live coding demonstration helped them learn about programming concepts and the demonstration improved their learning. While students were less likely to strongly agree that the visualizations helped them learn how to program, students were more likely to strongly agree that the demonstration and visualizations helped them learn about programming. The responses to the two open questions at the end of the survey reflected that students mostly liked the visualizations, but some felt that too many were included in the live coding demonstration. The results of the survey, summarized in Figure 3, show that students agreed that live coding using Process Feedback is a useful resource for educators and suggest that the incorporation of the tool is a means of encouraging students to engage in higher order thinking skills as students think about the process of programming.

Students are Comfortable Showing Their Process

We were delighted to find that all of the students interviewed unanimously agreed that they would be comfortable showing their programming process to their instructors. Students were not concerned that they would feel monitored and overwhelmingly agreed that showing their process to instructors would be beneficial to their learning. Student #3, who was not enrolled in a course that used Process Feedback, noted that their friends who were required to use Process Feedback for their assignments were discussing their process reports and showed their reports with student #3, without asking. This result corroborates findings from a 2023 study on plagiarism deterrence in an introductory computer

science course, where only 9.5% of students surveyed felt anxious about submitting a history of their keystrokes along with their assignment to their professor (Hart et al., 2023).

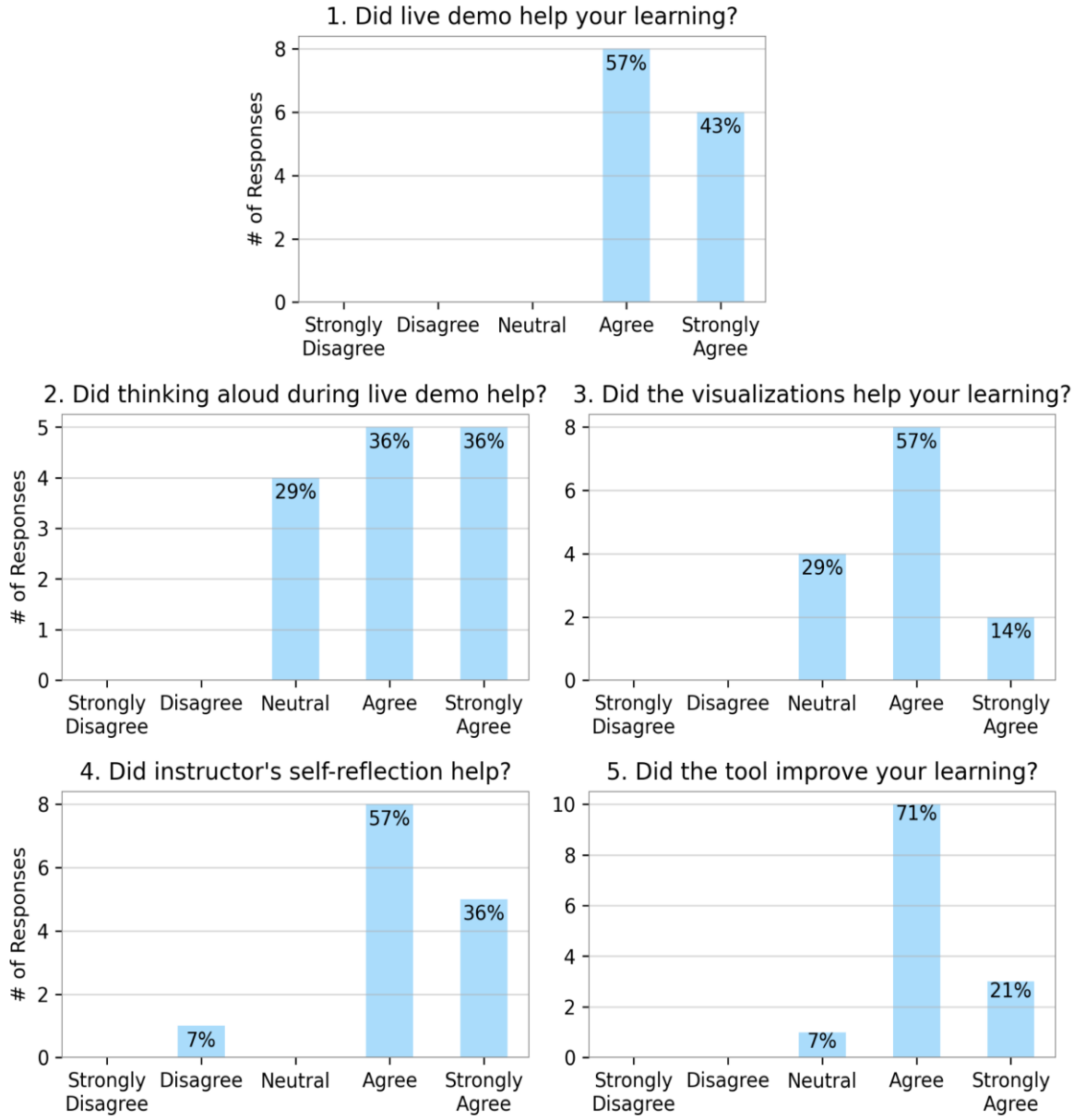

Figure 3. Summary of fourteen students' responses to the five questions asked after the live coding demo using the Process Feedback tool.

There was an exception though. Instructor #3 noted that, despite several reminders and repeated instructions, there were a few students in the class who always submitted plain C++ source code files instead of Process Reports. The instructor further added that these were students who had previously turned in unoriginal work. According to the instructor, the reluctance of these students towards using Process Feedback confirmed the assumption that students who complete the assignment as intended would not have any discomfort showing their process, but students who turn in unoriginal work would be uncomfortable showing their process. Students interviewed shared that they were comfortable showing their process because their instructor clearly explained to them the

value of process-aware coding. They found the ideas of sharing their process motivating because they knew that showing their process could improve their grades or that they would receive appreciation and encouragement from their instructors. As Turner and Paris argue in the context of children's literacy, choice is a "powerful motivator" that not only provides meaning to tasks, but encourages personal responsibility for learning (Turner & Paris, 1995). The results suggest that by using process reports to recognize the unique choices students make in approaching a programming problem, educators can empower students to take responsibility for their work and motivate them to learn.

Easy Visualizations are Helpful for Students and Instructors

Overall, professors and students alike found the visualizations in the process report to be helpful. While some students and instructors had personal preferences, only one visualization was generally found to be overwhelming and difficult to interpret, and that was the active lines evolution chart. Because of the large amount of data displayed and information conveyed by this chart, students and instructors struggled to recall how they interpreted the visualization or ignored it altogether. After receiving an explanation of the visualization during the interviews, Instructors #1 and #2 and Students #1, #2, and #3 changed their minds and stated that the visualization would be helpful for instructors to understand students' processes. All students interviewed found the Active Time and Thinking bar chart easiest to comprehend, and they expressed that seeing the thinking time gaps made them feel that they were not productive enough, which motivated them to invest more effort.

Instructors Find Process Reports Insightful and Easy

All the instructors who used the tool believed that it allowed them to better understand student programming processes, including when students succeeded or struggled in their work. Likewise, students unanimously thought that if they showed their process reports to an instructor, they would be able to receive improved feedback on their work and programming process. Instructor #2 asserted that seeing students' process reports allowed them to identify whether the student was learning and understanding the meaning of the assignment, and Instructor #3 stated that looking at the process reports showed whether students understood the concepts they were asked to apply in their assignment. All of the instructors noted that they were able to quickly determine the amount of effort that students put into their work and whether they had copied and pasted unoriginal content. In these instances, instructors looked at the combination of typing speed and amount of text added, and then used the Revision Version Comparison feature to discern how much text was added at a given time point. Even when a student copied line-by-line, Instructor #3 could identify concerns with the student's process due to the prolonged steadiness of the typing speed, linearity of additions, and short active duration.

While estimates suggest that the rate of cheating in university courses ranges between 40-96%, only 1.3% of cases are detected, and plagiarism is particularly common in introductory computer science courses due to the similarity of correct solutions (Fraser, 2014). Given the ease with which Process Feedback enables instructors to assess effort and identify (potentially) unoriginal work, it is evident that process-oriented technology presents a promising alternative to existing auto-grading and plagiarism detection technologies.

Hesitance to Use Because of Specific Desired Features

When asked to implement Process Feedback in their courses, Instructors #1 and #3 expressed some hesitation. Despite any initial uncertainty, all instructors found Process Feedback easy to incorporate into their teaching plan. Students also showed hesitancy, as both Instructors #1 and #2 reported that students preferred other development environments for the completion of their

programming assignments. Instructor #1 cited the editor's lack of error highlighting and slightly longer execution time as reasons why students would prefer to code in another environment. Student #4, who was enrolled in Instructor #1's course, expressed a preference for working in Visual Studio Code because it provides error highlighting and automatic code completion features.

A common concern among some instructors and students in higher-level courses was the tool's inability to support advanced libraries. While Process Feedback supported built-in libraries for languages like Python and C++, it did not allow students to upload custom libraries. For instance, Student #1 was unable to import a library for pseudo-random number generation. Instructor #1 shared that in cases where including similar dependencies would be necessary, such as in a 'pom.xml' file for Java projects, students would need to use a different development environment for their work. However, all interviewees agreed that these technical issues would not impact the use of such a tool in introductory courses, such as Instructor #3's course, where students were learning the fundamentals of programming. All the instructors who were interviewed stated that they were planning to use the tool in future courses for individual programming assignments, and Instructor #3 continued using the tool in multiple classes for two semesters beyond the duration of this study.

Limitations of the Research

Since this study lacked a control group and was not designed as an experimental study, its results are not meant to be replicable or generalizable. The case studies were conducted over a single summer semester at one university, and not all assignments utilized Process Feedback, which necessitates further research to evaluate the tool's effectiveness over a more extended period an in various contexts to confirm the durability of the findings. To overcome these limitations, future studies should investigate the tool's causal impact by using it for all evaluations in a programming course and by establishing a control group to assess differential learning outcomes.

Furthermore, potential bias may have been introduced by the involvement of the first and third authors in developing Process Feedback and that only volunteering students were interviewed. These factors could have influenced the perceived sentiment of students towards the tool and affected the authors' evaluation of its effectiveness. To address these limitations, future studies could involve a larger, more representative sample of students for qualitative and quantitative analysis of their perceptions and experiences with the tool with researchers maintaining a more neutral stance.

In addition to studying the use of the tool in individual assignments, inquiry should be made into the usefulness of the tool in peer-to-peer feedback. A 2010 study had found that students who received training in peer assessment skills outperformed peers who did not receive such training (Xiao, 2010). To analyze the impact of Process Feedback on peer assessment, educators should instruct students to share process reports with each other and gain insight from their work. Furthermore, the impact of the tool on group programming assignments should be evaluated to determine how the tool can identify contributions and efforts from individual group members.

4. CONCLUSION

This study explored instructors' and students' perceptions of the effectiveness of using Process Feedback, a free online compiler for process-oriented coding, in addressing current issues in programming education, most notably the challenges of assessing students' work for correctness and originality. The survey showed that students found the use of Process Feedback in a live coding demonstration helpful for learning about the coding process. Qualitative interviews revealed that students found the tool beneficial for their learning and for instructors' assessment of their coding assignments. Similarly, instructors reported that they could effectively determine the effort invested in

and the originality of their students' work. These results suggest that process-oriented technologies like Process Feedback successfully mitigate many of the limitations of current teaching and learning tools in coding. By utilizing such technologies in the classroom, teachers can enhance metacognitive teaching strategies like live coding and promote student motivation. Furthermore, our findings imply that process-oriented technology can assist educators in identifying plagiarism, a particularly challenging task in programming education. We believe these technologies hold significant potential for application beyond programming education, across various academic disciplines.

Acknowledgments

We would like to thank Sameep Dhakal and Subodh Dahal at Tribhuvan University for helping us analyze the data. We would also like to thank Aadya Jha and Abhigya Singh at Delhi University for providing useful comments to improve the quality of the manuscript.

5. REFERENCES

Adhikari, B. (2023). Thinking beyond chatbotsrsquo; threat to education: Visualizations to elucidate the writing or coding process. *Education Sciences*, 13(9).

Cutler, L. & Graham, S. (2008). Primary Grade Writing Instruction: A National Survey. *Journal of Educational Psychology*, Vol. 100, No. 4, 907–919.

Dennen, V. P. & Burner, K. J. (2008). The cognitive apprenticeship model in educational practice. In *Handbook of research on educational communications and technology*, pages 425–439. Routledge.

Devore-McDonald, B. & Berger, E. D. (2020). Mossad: Defeating software plagiarism detection. *Proceedings of the ACM on Programming Languages*, 4(OOPSLA):1–28.

Fraser, R. (2014). Collaboration, Collusion and Plagiarism in Computer Science Coursework. *Informatics in Education*, 13(2): 179-195.

Graham, S., Harris, K. R., & Chorzempa, B. F. (2002). Contribution of spelling instruction to the spelling, writing, and reading of poor spellers. *Journal of Educational Psychology*, 94(4), 669–686.

Graham, S. & Perin, D. (2007). A meta-analysis of writing instruction for adolescent students. *Journal of educational psychology*, 99(3):445.

Graham, S., & Sandmel, K. (2011). The process writing approach: A meta-analysis. *The Journal of Educational Research*, 104(6), 396-407.

Grønli, T. & Fagernes, S. (2020). The live programming lecturing technique: A study of the student experience in introductory and advanced programming courses. In *Norsk IKT-konferanse for forskning og utdanning*, volume 4.

Hart, K., Mano, C., & Edwards, J. (2023). Plagiarism Deterrence in CS1 Through Keystroke Data. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1 (SIGCSE 2023)*.

Hayes, J. R. (1996). A new framework for understanding cognition and affect in writing. In C. M. Levy & S. Ransdell (Eds.), *The science of writing: Theories, methods, individual differences, and applications* (pp. 1–27).

Hayes, P. J. & Mouradian, G. V. (1981). Flexible parsing. *American Journal of Computational Linguistics*, 7(4):232–242.

Hristova, M., Misra, A., Rutter, M., & Mercuri, R. (2003). Identifying and correcting java programming errors for introductory computer science students. *ACM Sigcse Bulletin*, 35(1):153–156.

Kölling, M. & Barnes, D. J. (2004). Enhancing apprentice-based learning of java. In *Proceedings of the 35th SIGCSE technical symposium on Computer science education*, pages 286–290.

Kustanto, C. & Liem, I. (2009). Automatic source code plagiarism detection. In *2009 10th ACIS International conference on software engineering, artificial intelligences, networking and parallel/distributed computing*, pages 481–486. IEEE.

Livingston, J. A. (2003). *Metacognition: An Overview*. ERIC.

Marceau, G., Fisler, K., & Krishnamurthi, S. (2011). Mind your language: on novices' interactions with error messages. In *Proceedings of the 10th SIGPLAN symposium on New ideas, new paradigms, and reflections on programming and software*, pages 3–18.

Mekterović, I., Brkić, L., & Horvat, M. (2023). Scaling automated programming assessment systems. *Electronics*, 12(4):942.

Nienaltowski, M., Pedroni, M., & Meyer, B. (2008). Compiler error messages: What can help novices? In *Proceedings of the 39th SIGCSE technical symposium on Computer science education*, pages 168–172.

Prather, J., Pettit, R., McMurry, K. H., Peters, A., Homer, J., Simone, N., & Cohen, M. (2017). On novices' interaction with compiler error messages: A human factors approach. In *Proceedings of the 2017 ACM Conference on International Computing Education Research*, pages 74–82.

Pritchard, R. & Honeycutt, R. (2007). Best Practices in Implementing a Process Approach to Teaching Writing. In *Best Practices in Writing Instruction*, pages 28-49.

Rowley, J. (2014). Designing and using research questionnaires. *Management research review*, 37(3):308–330.

Rubin, M. J. (2013). The effectiveness of live-coding to teach introductory programming. In *Proceeding of the 44th ACM technical symposium on Computer science education*, pages 651–656.

Schober, M. F. & Conrad, F. G. (1997). Does conversational interviewing reduce survey measurement error? *Public opinion quarterly*, pages 576–602.

Schultz, K. & Fecho, B. (2000). Society's child: Social context and writing development. *Educational Psychologist*, 35(1), 51–62.

Selvaraj, A., Zhang, E., Porter, L., & Raj, A. G. S. (2021). Live coding: A review of the literature. In *Proceedings of the 26th ACM Conference on Innovation and Technology in Computer Science Education* V. 1, pages 164–170.

Shaffer, D. W. & Resnick, M. (1999). "Thick" authenticity: New media and authentic learning. *Journal of interactive learning research*, 10(2):195–216.

Turner, J. & Paris, S. G. (1995). How literacy tasks influence children's motivation for literacy. *The Reading Teacher*, 48, 662-673.

Ullah, Z., Lajis, A., Jamjoom, M., Altalhi, A., Al-Ghamdi, A., & Saleem, F. (2018). The effect of automatic assessment on novice programming: Strengths and limitations of existing systems. *Computer Applications in Engineering Education*, 26(6):2328–2341.

Yadav, A., Burkhart, D., Moix, D., Snow, E., Bandaru, P., & Clayborn, L. (2015). Sowing the seeds: A landscape study on assessment in secondary computer science education. *Comp. Sci. Teachers Assn.*, NY, NY.

Xiao, Y. (2010). The effects of training in peer assessment on university students' writing performance and peer assessment quality in an online environment. Doctor of Philosophy (PhD), Dissertation, Teaching and Learning, Old Dominion University.