ASEE 2022 ANNUAL CONFERENCE
Excellence Through Diversity
MINNEAPOLIS, MINNESOTA, JUNE 26TH-29TH, 2022

Paper ID #36757

Work In Progress: CodeCapture: A Tool to Attain Insight into the Programming Development Process

Naman Gulati

Angy Higgy

Hamid S Timorabadi (Assistant Professor, Teaching Stream)

Hamid Timorabadi received his BSc, MASc, and PhD degrees in Electrical Engineering from the University of Toronto. He has worked as a project, design, and test engineer as well as a consultant to industry. His research interests include the application of digital signal processing in energy systems and computer networks. He also has deep interest in engineering education and the use of technology to advance the learning experience of undergraduate students.

Work In Progress: CodeCapture: A Tool to Attain Insight into the Programming Development Process

Abstract

Most introductory computer programming courses focus on giving students a good understanding of fundamentals and programmatic problem solving. Programming assignments are a way for students to reinforce these concepts and for instructors to evaluate students' comprehensions. In a typical assignment, students are given a problem to which they develop a programmatic solution and submit its final iteration for evaluation. Instructors only see this static and final submission, and the decision making or the process of problem solving that led to the final submission is not captured. This paper presents a tool called CodeCapture that periodically captures snapshots of a student's programming assignment source code over the development period and provides insight into their problem-solving process. Instructors can leverage data provided by this tool to identify areas that students have difficulty. Subsequently, instructors can give insightful feedback to students via a combination of metrics describing students' development processes, even in a large-scale classroom. In this paper, a discussion on how CodeCapture overcomes gaps in the existing tools is presented. The technical specifications and the trial results from where CodeCapture was able to enrich the feedback to students regarding their assignments is also presented.

1. Introduction

Most computer programming courses focus on teaching teach fundamental concepts and programmatic problem solving. Problem solving is considered as the *process* of working through ambiguity to achieve a certain goal [1]. Assignments in introductory courses is viewed as a dominant tool in reinforcing theoretical concepts and assessing students' problem-solving ability [2]. In a typical assignment, students are given a problem to which they program a solution and submit the final iteration of their code for evaluation. The student's entire *process* of solving problems, interacting with the concepts and reinforcement of theory is summarized into this final submission. Providing valuable feedback is critical to enriching a student's problem-solving skills [3]. However, it is an understandably challenging task to provide elaborate feedback on a student's methodology when an instructor only receives one final submission of the assignment in the entire process [4, 5].

Another large obstacle that makes it challenging to provide quality feedback is the limited amount of time available from instructors [6]. Introductory programming courses can have over 400 students. Reviewing each student's code and extrapolating their approach to the solution from their final iteration for such a class size can take a considerable amount of time [7]. As a result, most classrooms make use of Automated Assessment Tools (AATs) that predominantly test students' submissions against a fixed set of test cases to gauge for functional accuracy [8]. Using these tools as the sole source of feedback imposes challenges in building students' metacognitive awareness which is critical for their own ability to consciously identify and refine their problem-solving processes [9].

CodeCapture is a tool that assists instructors to quickly provide feedback on students' problem solving and development processes, even before the student submits their assignment. CodeCapture periodically analyzes a student's code as it is being developed. CodeCapture periodically analyzes a student's code as it is being developed and shows instructors a progression view of the code's complexity, implementation effort, and accuracy.

In this paper, Section 2 identifies related works and similar projects that we used as a functional benchmark and Section 3 describes our objectives in establishing CodeCapture. Subsequently, the CodeCapture system: the classroom implementation process and the technical details specific to the design of CodeCapture is presented in Section 4. Finally, Section 5 describes the study results from a classroom reporting positive results from students.

2. Related Work

A significant amount of technological innovation directed at introductory programming classrooms are variations of Automated Assessment Tools (AATs) with the core functionality of testing student's code for accuracy [10, 11]. Nevertheless, more recently, a study by Pettit et al. showed that analysis of iterative improvements in programming assignments can help refine programming instruction [5]. This study highlighted that implementation of more dynamic testing of submissions which analyzes code style and complexity can help identify students' gaps in understanding. A study by Ala-Mutka et al. showed how feedback on programming style helped students' programs become more "understandable and reliable" [12]. CodeCaputre presents instructors with graphs of metrics describing logical efficiency (complexity), development style, and functional accuracy.

Pensieve is a study that displays raw snapshots of a student's code as they progress through an assignment [13]. A similar project by Novák et al. discusses a system that tracks students' continuous workflow over a semester of learning [14]. Pensieve is a tool that is designed to promote metacognitive discussions between students and instructors on students' problem solving. It allows instructors to analyze the progression of the student's work over automated snapshots that are taken as the student works on an assignment. This tool automatically captures and directly presents raw snapshots to instructors; it is designed to be used for 5-10 minute instructor - student interactions, where the snapshots are reviewed [13]. The project pursued by

Novák et al. focused on tracking students' workflow in programming assignments for obtaining insights on their implementation methodologies [14]. The architecture of the system involves a remote Git repository that receives code snapshots each time a student runs their program. At the end of the assignment, instructors receive a statistical analysis of a student's work over time displayed in the form of graphs. These graphs aggregate the information captured on the student over the tracking period and thus are quicker to interpret than raw snapshots. The following metrics are displayed to instructors [14].

- **Code Activity -** The number of lines of code added over a specific period of time.
- **Time Activity -** The total time spent working on the assignment.
- **Frequencies -** Hours and days when the code activity is most frequent.

Analyzing these two projects, we found that there were shortcomings, which are addressed in CodeCapture. Pensieve was designed for 5-10 minute instructor - student interactions for feedback discussion. Spending 5-10 minutes per student in large classrooms is often infeasible [6]. Pensieve can be difficult to adopt in a classroom as it would demand significantly more time from instructors than what they currently spend and may not inspire instructors to employ the tool. The project pursued by Novák et al. requires a student to run their program for changes to be pushed to the repository. A student may not run their program until the last few days of the assignment period, this causes the tool to omit a lot of details about students' problem-solving and exploration of the solution space. Additionally, the data is presented to the instructors at the end of the assignment period, not allowing them to see how their students are doing while they are working on the assignment. CodeCapture addresses these concerns by automatically capturing student's code throughout the development process and uses it to generate concise graphs, while performing independently from the student's actions to avoid impeding their development process.

3. Key Objectives

CodeCapture was designed with the following objectives, which formed the basis of our evaluation in a classroom trial of CodeCapture.

- **Improve quality of the feedback provided by instructors**

In most cases programming assignments are solely evaluated based on code correctness, typically assessed using Automated Assessment Tools (AATs). The feedback students receive consists of their score from the AATs as well as comments from course staff on code style. Instructors do not have any means to identify which concepts students had difficulties while attempting the assignment. CodeCapture supplements instructors with additional insight from every step and areas that student has difficulty to understand. For instance, an instructor can identify a specific topic that all students or one student spend the most time to solve. Then, the instructor can provide supplementary material or further explain that certain topic during lectures.

- **Provide instructors the ability to identify a student’s development process**

The goal of assignments in introductory programming classes is to not only teach students basic computer science concepts, but also to teach them the process of arriving at a solution using an effective development process. With current evaluation methods, a final submission can identify a student’s level of understanding of the concept being taught in the assignment. However, it does not provide insight to how the student arrived at the solution.

- **Maintain the quality of a student’s workflow**

CodeCapture should be a tool that provides benefits to instructors without compromising the quality of a student’s workflow during an assignment. In particular, the tool should not interfere with the student’s work in any way that would disrupt their progress or workflow.

4. CodeCapture System

As a student works on an assignment, CodeCapture periodically takes snapshots of all files that are updated. These snapshots are analyzed for complexity, accuracy, and development effort on a function-by-function granularity. The complexity metric indicates the cyclomatic complexity of the code, describing the number of linearly independent execution paths in each function [15]. When creating an assignment in CodeCapture, instructors have a provision to upload a Test Package. This package contains tests which are run against every captured snapshot of student code, generating pass/fail results to comprise the accuracy metric. Development effort shows how much incremental code was developed throughout the snapshots, giving insight to the speed of development.

5. Classroom Use

CodeCapture system has two interfaces: a command-line tool (CodeCaptureCLI) that students interact with and a web interface (CodeCaptureWebUI). The web interface allows an instructor to: create assignments and view students’ snapshot metrics.

1) An instructor first creates an assignment through the web dashboard. In doing so the instructor specifies a start date and end date for the assignment, and uploads a zip archive containing the Tests Package.
2) Once the assignment starts, CodeCapture Snapshot System (that is installed with the CLI in student development computers) starts polling for updates to assignment source code. This allows students to work on their assignments seamlessly without having to interact with the CodeCapture system.
3) An instructor can view the metrics on a student’s development process and along with class-average values regarding the same metrics. The instructor can then employ these metrics to modify instruction to the class or offer feedback to an individual student.

4) After the assignment end date students' assignment source code is no longer monitored.

6. Implementation

As shown in Figure 1, CodeCapture is comprised of three high level systems: student facing CodeCapture CLI, Insights System which is responsible for analyzing code, and the WebUI. These three systems together allow for automated periodic snapshotting, analysis, and display of student code and relevant metrics. The analytics are performed by the Insights Engine, running on a server. This offloads the computation to the server and reduces impacts on student's computer.

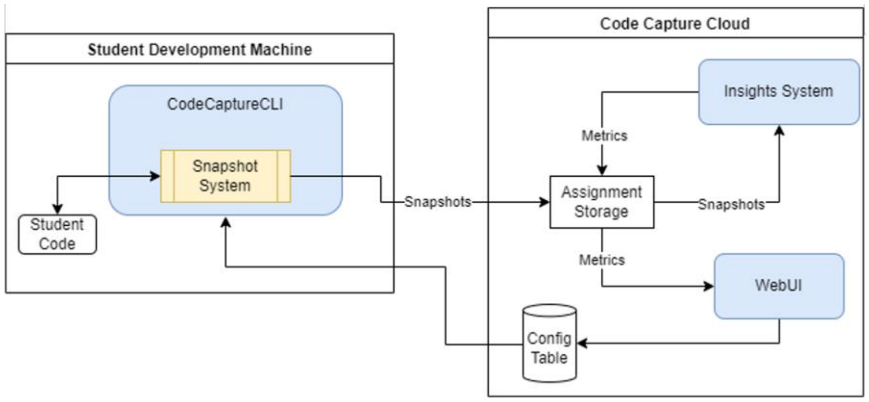
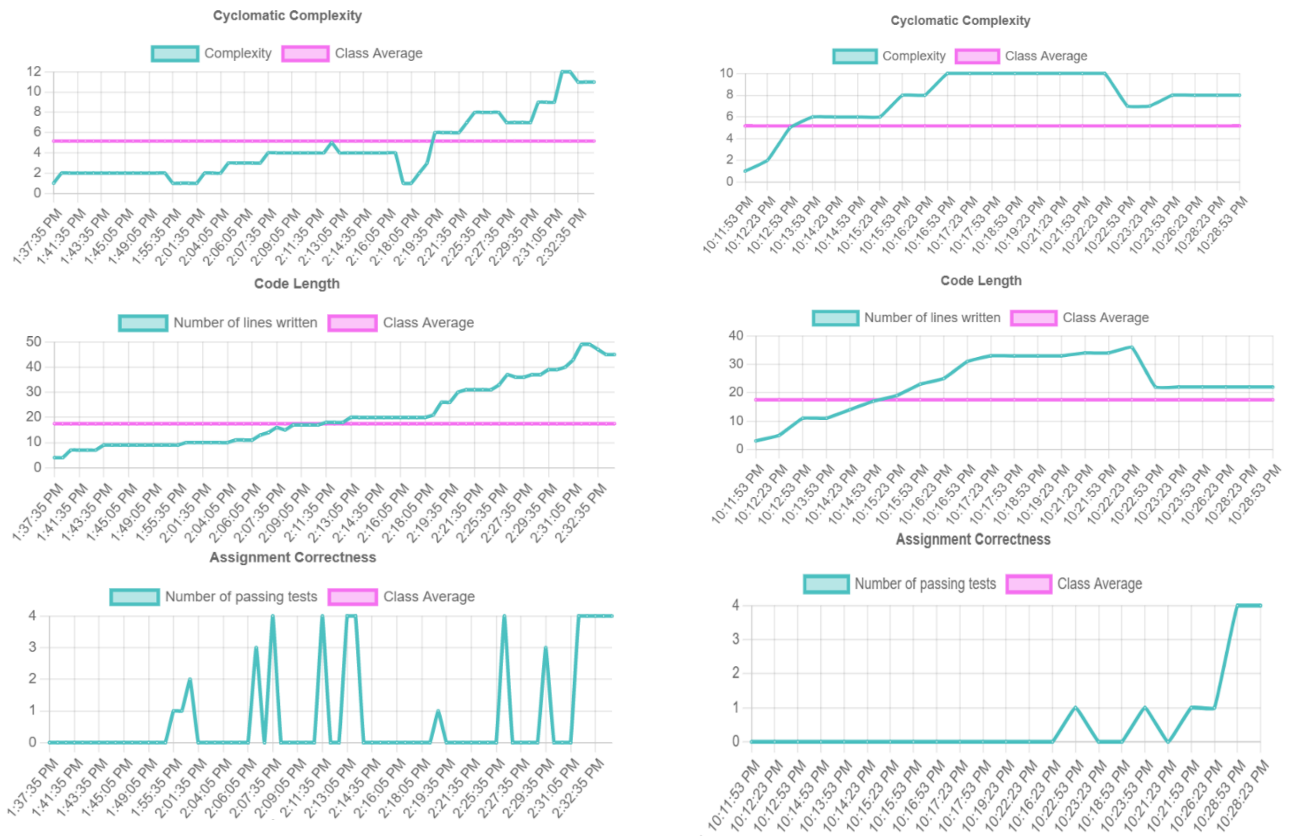


Figure 1: CodeCapture system architecture.

The three high level system architecture of CodeCapture are described as follows.

1) *CodeCapture WebUI* allows an instructor to set up assignments and view analytics for students working on assignments. The instructor must enter the assignment's name, start date, end date, and optionally a zip archive containing a test package (used to test student code). The test archive is uploaded to a file storage system that also is used to save snapshots and data for metrics generated and is referred to as Assignment-Storage. The assignment configuration is inserted into a database table named Config-Table. As students work on their assignment instructors can view the complexity, accuracy, and effort metrics of students' codes. The WebUI retrieves metrics data stored in Assignment-Storage and uses it to render the graphs, as shown in Figure 2.



a) Student A: Time spent 34 minutes and 30 seconds. **b)** Student B: Time spent 11 minutes and 30 seconds.

Figure 2: CodeCapture assignment metrics for two students.

2) *CodeCapture CLI* is students' interface to CodeCapture. Once configured by the student, it periodically checks for new assignments assigned to them from Config-Table. On the start date of the assignment a daemon "snapshot system" process is launched that polls for file update events in the assignment source code directory. Upon each file update detected, a copy of the updated file is stored in Assignment-Storage and the Insights System is notified, along with this an archive of the entire source code is also temporarily uploaded to the Insights System. The source code archive is used to build and test the source code for the accuracy metric. When the assignment is scheduled to end, the "snapshot system" terminates.

3) *CodeCapture Insights System* is responsible for analyzing snapshots and computing the complexity, accuracy, and development effort metrics. It receives a new snapshot through Assignment-Storage and uses Lizard, an open-source tool, to analyze the snapshot [16]. For each function in the captured file, Lizard provides the cyclomatic complexity and length in lines of code. These two values serve as the basis for incremental complexity and development effort over snapshots, shown to instructors through the WebUI. To compute the accuracy metric, the source code archive attached to each snapshot is extracted and tested using the test package provided by the instructor. The Insights System stores the complexity,

effort and test results to a file containing all data points for each function analyzed, in Assignment-Storage.

7. Testing and Results

The first phase of testing involved a beta test to assess the results against the key objectives that were outlined. Two problems were selected and assigned to a group of five Computer Engineering students at the University of Toronto with a strong background in computer programming. These problems required the implementation of common programming concepts such as loops, recursion, and tree traversals. Each student was asked to set up CodeCapture and allow it to monitor their development process as they solved the two questions. After they finished their assignment, the metrics reported for each student by CodeCapture were shown to an instructor who used the data to provide feedback to students.
Following their completion of the assignment, the students were asked to complete a short survey on their experience using CodeCapture. The questions were designed to gain insight into the tool's ease of use, the impact it had on student's dedication to the assignment and to compare the quality of feedback received with CodeCapture to the feedback they typically receive from instructors. The questions required students to give a discrete response, with the response types being a combination of multiple-choice selections and Likert scale ratings.

The following results were drawn from the questions where students were asked to provide a Likert scale rating ranging from 0 – 5:

- Students were asked to provide a Likert scale rating of "how feedback given on this assignment compared to feedback they received in the past on programming assignments". Students found feedback from CodeCapture data to be considerably more useful ($\mu = 4.2$).
- All students rated CodeCapture's set-up process as either *'very easy'* (3/5) or *'easy'* (2/5). Four out of five students found no impediments in their development process from CodeCapture.

Metrics generated from two student's development processes on one of the assigned problems are shown in Figure 2. Metrics for Student A show slow development of code that was significantly longer and more complex than the rest of the students. Whereas the metrics for Student *B* show a polished and incremental progression in complexity and code additions over less than half the time taken by Student A. It became evident that Student *A* rushed into writing code and then spent a significant amount of time fixing/debugging to pass all the test cases. Whereas Student *B* solved the same problem by spending some time in planning and coming up with an algorithm prior to any coding attempts. This is evident through the correctness metric which shows that the student understood what the correct solution is and consistently passed all test cases once the solution was implemented. Student *A* was provided with the feedback that their code seemed to be developed through a lot of trial and a planning or analysis of the problem before development could have helped them reach an optimal solution quickly. Whereas Student *B* was given the feedback that their code seemed to be developed very methodically and in a well-planned fashion.

After a discussion with both students, Student *A* explained that they tried to solve the problem quickly and jumped straight into development; this led them to overlooking complexities and edge cases which they spent a lot of time addressing towards the end. Student *B* confirmed that they had looked at the problem the day before they started developing the solution and had planned out the approach which they were going to take. Both students' responses confirmed exactly the intuition about their respective development methods that the instructor had when looking at the CodeCapture metrics.

We will be repeating this study on a larger scale introductory computer science classroom to evaluate the impact of code capture on a more representative classroom.

References

[1] M. E. Martinez, “What Is Problem Solving?,” vol. 79, no. 8, pp. 605–609, 1998, [Online]. Available: http://www.jstor.org/stable/20439287.

[2] D. R. Woods, T. Kourti, P. E. Wood, H. Sheardown, C. M. Crowe, and J. M. Dickson, “Assessing Problem-Solving Skills: Part 1. The Context for Assessment,” vol. 35, no. 4, pp. 300–307, 2001.

[3] S. A. Bjorklund, J. M. Parente, and D. Sathianathan, “Effects of faculty interaction and feedback on gains in student skills,” vol. 93, no. 2, pp. 153–160, 2004.

[4] F. M. Van der Kleij, R. C. W. Feskens, and T. J. H. M. Eggen, “Effects of Feedback in a Computer-Based Learning Environment on Students’ Learning Outcomes: A Meta-Analysis,” vol. 85, no. 4, pp. 475–511, 2015, [Online]. Available: http://www.jstor.org/stable/24753021.

[5] R. Pettit, J. Homer, R. Gee, S. Mengel, and A. Starbuck, “An empirical study of iterative improvement in programming assignments,” 2015, pp. 410–415.

[6] K. Ala-Mutka and H.-M. Jarvinen, “Assessment process for programming assignments,” 2004, pp. 181–185.

[7] Q. Hao et al., “Towards understanding the effective design of automated formative feedback for programming assignments,” pp. 1–23, 2021, doi: 10.1080/08993408.2020.1860408.

[8] K. Ala-Mutka, “A Survey of Automated Assessment Approaches for Programming Assignments,” vol. 15, no. 2, pp. 83–102, 2005, doi: 10.1080/08993400500150747.

[9] J. Prather, R. Pettit, K. McMurry, A. Peters, J. Homer, and M. Cohen, “Metacognitive difficulties faced by novice programmers in automated assessment tools,” 2018, pp. 41–50.

[10] P. Ihantola, T. Ahoniemi, V. Karavirta, and O. Seppälä, “Review of recent systems for automatic assessment of programming assignments,” 2010, pp. 86–93.

[11] J. C. Caiza and J. M. del Álamo Ramiro, “Programming assignments automatic grading: review of tools and implementations,” 2013.

[12] K. Ala-Mutka, T. Uimonen, and H.-M. Jarvinen, “Supporting students in C programming courses with automatic program style assessment,” vol. 3, no. 1, pp. 245–262, 2004.

[13] L. Yan, A. Hu, and C. Piech, “Pensieve: Feedback on coding process for novices,” 2019, pp. 253–259.

[14] M. Novak, M. Biňas, M. Michalko, and F. Jakab, “Student’s progress tracking on programming assignments,” 2012, pp. 279–282.

[15] T. J. McCabe, “A complexity measure,” no. 4, pp. 308–320, 1976.

[16] T. Yin, “Lizard.” https://github.com/terryyin/lizard [accessed Jan. 03, 2022].