Breaking the Textbook Paradigm: Increasing Access by Removing Words

Elise Deitrick, Codio

Elise has a BS in Computer Science and PhD in STEM Education. Her thesis was on interdisciplinary, collaborative computing using mixed methodologies. Elise combines her over a decade of teaching experience with her research background to create evidence-based computing education tools in her current role at Codio.

Maura Lyons, Codio

Maura is a Marketing Associate at Codio with a BA in Psychology and English.

Mr. Joshua Richard Coughlin Stowell Ball, Codio

Joshua Ball is Codio's Vice President of Marketing and a Senior Fellow at the National Institute for Deterrence Studies.

Breaking the Textbook Paradigm: Increasing Access by Removing Words

Abstract

Textbooks are an anachronistic element of education in the 21st century which previous research shows students do not read despite reading assignments. For over a decade, computing education has evolved from textbooks to ebooks to interactive learning experiences with animations, built-in IDEs, and autograders. More recent work has shown that many of these innovations such as paired programming, code visualizers, and Parsons problems have positive educational outcomes such as student engagement, retention, and increased learning gains – particularly for students from historically marginalized communities. Unfortunately, due to the origins of these learning experiences being textbooks, they often still look like text-heavy ebooks instead of evidence-based, interactive learning experiences.

This paper uses data from a computing education platform used in dozens of US universities and thousands of students to explore the relationship between how much time students spend reading compared to the amount of text in the reading assignment. Specifically, we do multiple regression analyses to understand how independent variables including word count, assessments, and whether students engaged with code, affect dependent variables such as assignment grade and how student's time was spent.

We find that even without fine-grained details, learning experiences where students are spending more time actively coding as opposed to reading result in higher performance. These at scale results solidify that it is time for the field to break out of the overly passive textbook paradigm and embrace learning experiences which center student opportunities to code.

Introduction

Recent research indicates a decline in engagement with traditional textbooks among post-secondary students in computing disciplines. Studies by Margulieux and Catrambone [1], and Amelink et al. [2], have highlighted a growing preference for interactive and digitally accessible materials over conventional textbooks. This shift is attributed to the digital-native characteristics of contemporary students, who favor multimedia-rich and interactive content [3]. Further, the study "Towards Modeling Student Engagement with Interactive Computing Textbooks" [4] illuminates this trend by demonstrating that interactive textbooks, particularly those utilizing Jupyter notebooks, significantly enhance student engagement through active code execution and modification.

The transition towards digital learning platforms has introduced various interactive elements designed to improve learning outcomes in computing education. Key features identified as particularly effective include immediate feedback, micro-assessments, Parsons problems, and

interactive simulations [5] - [8]. Supporting this, Sigarchian et al. [9], in their work on "Hybrid e-TextBooks as Comprehensive Interactive Learning Environments," emphasize the role of hybrid e-textbooks that combine digital publishing, interactive eBook design, and multimedia elements, showcasing a significant enhancement in learning outcomes.

The authorship of computing textbooks is evolving to include a mix of academic rigor and practical experience, with a noticeable trend toward involving industry professionals as co-authors or contributors [10]. This approach enriches educational materials with practical insights, essential in a field where technology changes rapidly. The case study at zyBooks, as discussed by Gordon, Lysecky, and Vahid [11], exemplifies this shift towards customizable content, highlighting the potential of digital platforms to offer updated, relevant, and engaging learning experiences.

Existing research highlights a significant change in how computing education is taught in post-secondary institutions. Shifting from the traditional textbooks to interactive and digital learning experiences aims to move away from text-heavy resources to more engaging and evidence-based interactive learning platforms. This change is expected to improve student engagement, retention, and learning outcomes by aligning educational tools with the modern world's technological advancements and today's students' learning preferences.

Background

Importantly, many recent educational innovations have the added benefit of increasing historically marginalized students' performance to be more inline with their more privileged peers. These innovations are varied and going through a selection of them provides an overview of the shape a potentially less traditional but more inclusive learning environment might take.

Active Learning Pedagogy in Computing

Active learning in computing provides opportunities for students to practice their skills and knowledge while learning rather than passively listening to a lesson. Two examples of such active learning include peer instruction and pair programming.

Peer instruction, as explored by Greer et al [12], highlights student-centered instruction, and swaps typical lecturing by moving information transfer out of and information assimilation into the classroom. When using peer instruction, students complete readings and practice before meeting as a class, and during class time, readings are discussed and more practices are completed [12]. Peer instruction has been shown to be effective in upper and lower level courses, improving student achievement, satisfaction, and self-efficacy. Peer instruction even improves retention rates in introductory level programming courses. Peer instruction is effective for multiple reasons. Firstly, the questions that replace lecturing are specifically designed to foster

interaction with course content. Secondly, students make use of classroom time to practice and ask their peers questions, actively engaging with material as opposed to passively listening to lectures.

Pair programming is the practice of two programmers sitting side-by-side on one computer -- one acting as the “driver” who controls the mouse and keyboard and the other acting as the “navigator” who watches for errors and makes suggestions -- switching roles regularly. Pair programming originally comes from a professional development style called extreme programming (XP), which has found many benefits in the practice, most notably in terms of code quality. “Pair programming was later adopted as a promising practice in higher education settings because of the benefits of increased retention of students continuing their study of computing, increases in programming confidence in students, and reductions in the ‘confidence gap’ between female and male students” [13]. As pair programming was further adopted, it was linked to success in introductory programming courses and increased satisfaction and enjoyment when programming [13].

Though some instructors worry about equal distribution of work, research has found that individual performance on exams is similar when using pair or solo programming [13]. Additionally, students might immediately push back against pair programming because they must practice soft skills that might not be typically associated with programming [14]. However, if instructors continue to use pair programming in their courses, students will manage the transition from solitary work to pair work, and they will benefit from it.

Active Learning Shifting Textbooks and Asynchronous Learning

Students had very limited access to information when textbooks first became an educational tool, so they needed to include anything deemed relevant to a given subject. However, as students now have access to more information than most libraries in their pockets, it no longer makes sense for students to lug around heavy, expensive physical textbooks.

Two overarching barriers—lack of access due to prohibitive costs and length—make traditional textbooks outdated, inefficient learning resources. For example, despite the excessive length of some texts, like the over 1000 pages of Cay Horstmann’s Big Java or the roughly 85,425 words of Think Python 2, research has shown that “94% of students spend less than two hours per reading assignment” [15]. That results in a lot of wasted pages and costs. Not only are students not reading, but many students are unable to even acquire the texts, as “65% of students at least once [do] not purchase a required textbook” [16].

However, making texts more interactive and less text heavy may resolve these issues. For example, minimizing text to the most critical points while using other representations, such as

images and tables to express details results in higher learning gains [15]. In digital modalities, instructors can use videos and other interactive elements to engage students in the material.

Research has also shown that less text (approximately 50% of the original) improves aspects of learning like completion time, amount remembered, and student satisfaction by 58% [17]. Shorter texts have also been shown to double student learning gains between pre and post-tests, and increase student satisfaction by 26% [15].

As you reduce text, instructors can replace passive reading with interactivity in several ways. There is, of course, adding practice exercises (whether they are graded or not), but instructors can also add visualizations and Parsons problems to help students think about code at a really low or really high level. One study, which measured student performance with interactive web-native content against the performance of students with static web content, found that "the average improvement score was 16% higher for participants given the interactive web-native content than the static web content" [18].

Lots of Computer Science is invisible, so using a visualization or simulation tool can help students "see" what is happening under the hood of their code. Abstract concepts like function calls, recursion, data structures, and scope can be concretized with a little animation. Previous research has found that students who interacted more with an earlier version of the Python Tutor code visualizer outside of class had statistically significant higher midterm exam grades. Additionally, using the visualizer outside of class correlated with the students' performance on the three unannounced quizzes [19].

Making textbooks less text heavy and including more interactive elements, like visualizers and practice exercises, benefits students' learning outcomes and makes learning more accessible by potentially reducing costs.

Coding and Skills Practice

Writing code is a time-consuming task for students. So, in addition to asking students to predict the output of existing code, Parsons problems offer another way to expose students to code without requiring large amounts of time. Research shows that while Parsons problems take "significantly less time than fixing code with errors or than writing the equivalent code... there was no statistically significant difference in the learning performance, or in student retention of the knowledge one week later" [20]. Additionally, as students make their way through units and semesters, the amount of work they do decreases, and yet, Parsons problems have been shown to be one of the most engaged with features on interactive platforms despite the drop off in student completion of tasks [21].

In computer science, we often ask students to build larger programming projects over the span of days or weeks. As teachers, we know that students do not always have the skills to project and time manage themselves well on these larger projects. Additionally, we know that trying to estimate how long it takes to plan, program, and test software projects is hard [22] and even software organizations in industry find it challenging to deliver software on time [23]. To help scaffold students on these larger projects, teachers often break projects up into milestones or separate gradable deliverables.

Benefits of Milestones in Programming Projects

By breaking these larger projects into distinct milestone assignments, teachers can more easily see where students are on the project, how much time each milestone takes, and how well students are performing on each milestone. For students, having distinct milestone assignments makes each piece more approachable and more clearly communicates expectations and deadlines.

Shaffer and Kazerouni [24] found in a third year Data Structures and Algorithms course that students who were given milestones "were more likely to finish their projects on time, produced projects with higher correctness, and finished the course with generally better outcomes". Additionally, within the set of students who were given milestones, "students who completed more milestones saw better outcomes."

Additional research shows that a variety of smaller problems increase student performance and reduce stress [25]. Using many small programs leads to students spending a sufficient amount of time on their work, and they do not wait until the last moment to begin their work [25].

There are no perceived downsides to adding milestones when comparing withdrawal rates and failure rates, and "an end-of-term survey indicated that student perceptions of the milestones were overwhelmingly positive" [24]. Breaking work into smaller increments not only benefits the teacher's understanding of how students are progressing, but also increases student's likelihood to complete work on time, with increased correctness within said work.

Methods

Data Collection

We collected University and College assignment data between March and December 2023 on the Codio platform. We pulled together the data fields listed and described in Table 1.

Table 1: List of data fields collected about each completed assignment

Data Field	**Description**
hashed_student_id	Anonymized student ID.
hashed_assignment_id	Anonymized assignment ID.
completed_date	Date the assignment was marked as completed by the student.
reading_word_count	A count of **student-facing** words in the guides feature. Words in assessment items were not included.
time_spent_seconds	Total time the student spent on the assignment as calculated by the platform in seconds.
coding_time_spent_seconds	Coding time or time the student spent typing was calculated by combining keystroke timestamps into sessions and summing all sessions associated with an assignment. A keystroke was considered the end of the session if there was no other keystroke within 10 minutes.
num_assessments	The number of assessment questions within the assignment.
answered_assessments	The number of assessment questions the student answered.

For the results below, a calculated data field which represented non-coding time was derived by subtracting the time coding from the total time spent.

The initial set of data included 684,507 completed assignments from 30,273 unique students. After filtering out rows where not all of the above data fields were able to be retrieved, we were left with 620,352 completed assignments from 28,229 unique students. Finally, we filtered out rows with flawed data including negative time spent, more time spent coding then total time spent, or grades above 100%. This left our final dataset as 525,941 completed assignments from 27,977 unique students.

Statistical Analysis

In the results section, linear regressions were run using the linreegress method from the Scipy library when comparing two variables. The regression equation, Pearson correlation coefficient, and Cohen's d are reported alongside a scatter plot.

Multiple linear regressions are run to test the relationship between multiple variables in predicting a single output variable. To accomplish this, we used OLS regression from statsmodel.api.

Data Context

To get a sense of the assignments, Table 2 provides descriptive statistics for the collected data fields. A number of assignments have a reading word count of 0. This could happen if there were only assessments which might occur for quizzes or exams. Comparing total time spent and coding time spent we see the average time spent coding is about a quarter of total time spent. The rest of that time was probably spent reading or answering assessments. In terms of assessments, most students completed all assessment questions within an assignment given how aligned number of assessments and answered assessments are. Finally, grades as reported on the platform for the completed assignments are very high with at least half of the grades being 100% and the average being 84%. This could be due to teachers setting up participation grading rules such that only completion and not correctness is taken into account.

Table 2: Descriptive statistics of collected data fields

	reading_ word_ count	**time_ spent_ seconds**	**coding_ time_ spent_ seconds**	**num_ assessments**	**answered_ assessments**	**grade**
mean	709.148	3,930.829	1,355.354	4.562	4.444	83.883
std	1,100.083	7,668.097	3,269.409	5.394	5.313	29.418
min	0	1	0	0	0	0
25%	0	788	128	1	1	80
50%	336	1,788	473	3	3	100
75%	941	3,952	1,331	6	6	100
max	15,332	906,172	174,000	111	96	100

Results

We start by exploring if the time spent, and more specifically how the time was spent, predicts the grades students earn on assignments (Figure 1). We found a statistically significant, but very

weak correlation between total time spent and the assignment grade ($p<0.05$ and $\rho = 0.043$). Interestingly, despite the weakness of the correlation, a medium effect size was found ($d=0.709$). Similar results were found for both non-coding and coding time - statistically significant but very weak correlations ($\rho = 0.024$ non-coding and $\rho = 0.059$ coding) with a medium effect size ($d=0.616$ non-coding and $d=0.550$ coding).

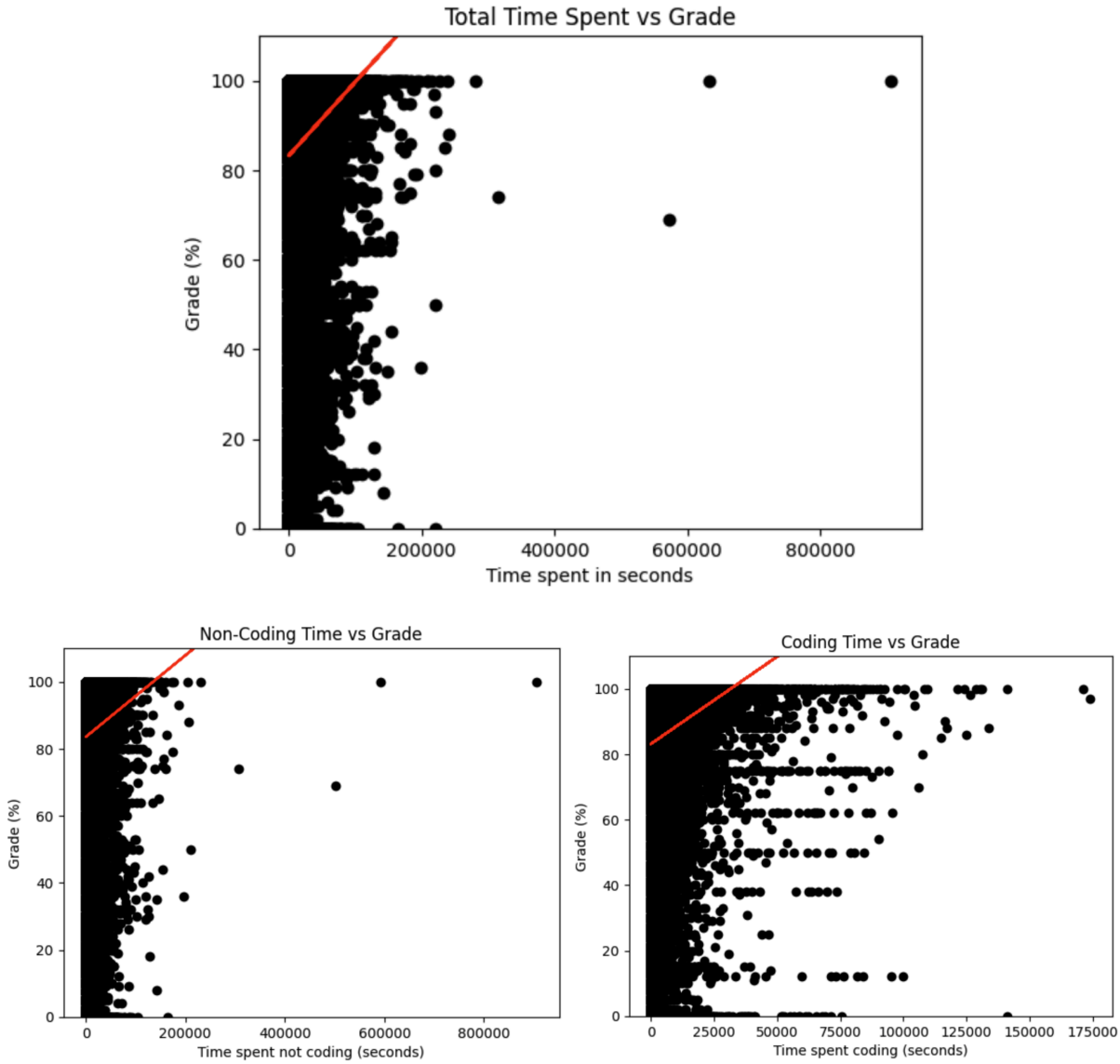

Figure 1: Types of Time Spent correlated with Grade
Total time spent vs Grade Regression line: $y = 0.000165x + 83.236$
Non-Coding Time vs Grade Regression line: $y = 0.000122x + 83.570$
Coding Time vs Grade Regression line: $y = 0.000533x + 83.160$

To try to understand if non-coding time is primarily related to reading, we explore the relationship between the number of words in the assignment and the ways students spent time in terms of coding or not (Figure 2). We found a statistically significant but weak correlation

between word count and total time spent ($p<0.05$ and $\rho = 0.043$). Despite the weak correlation, a medium effect size was found ($d=0.588$). Word count was correlated with both non-coding time and coding time weakly yet statistically significant ($\rho = 0.225$ non-coding and $\rho = 0.192$ coding) and with small effect sizes ($d=0.453$ non-coding and $d=0.265$ coding). A slightly stronger correlation and effect size are seen for non-coding time than coding time which one might expect, though often programming assignments can have rather lengthy specification documents which students must read and often re-read.

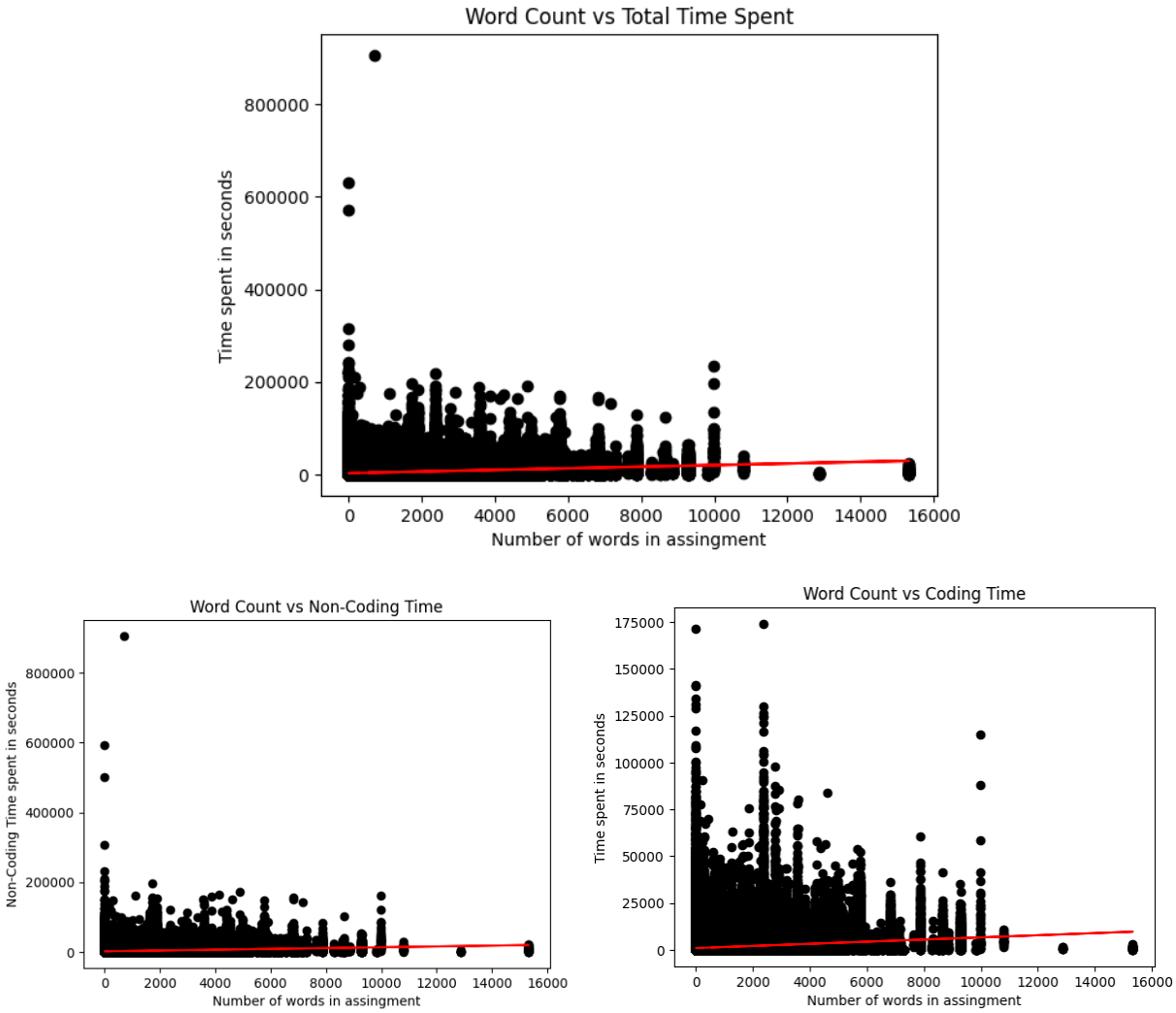

Figure 2: Word count correlated with types of Time Spent
Word count vs Total time spentRegression line: y = 1.743x + 2694.660
Word count vs Non-Coding Time Regression line: y = 1.173x + 1743.777
Word count vs Coding Time Regression line: y = 0.570x + 950.884

We also explored how answering assessment questions related to student time usage (Figure 3). All three regressions showed weak yet statistically significant correlations with medium effect

sizes (total: ρ = 0.204, p<0.05, d=0.724; non-coding: ρ = 0.176, p<0.05, d=0.635; coding: ρ = 0.2170, p<0.05, d=0.584).

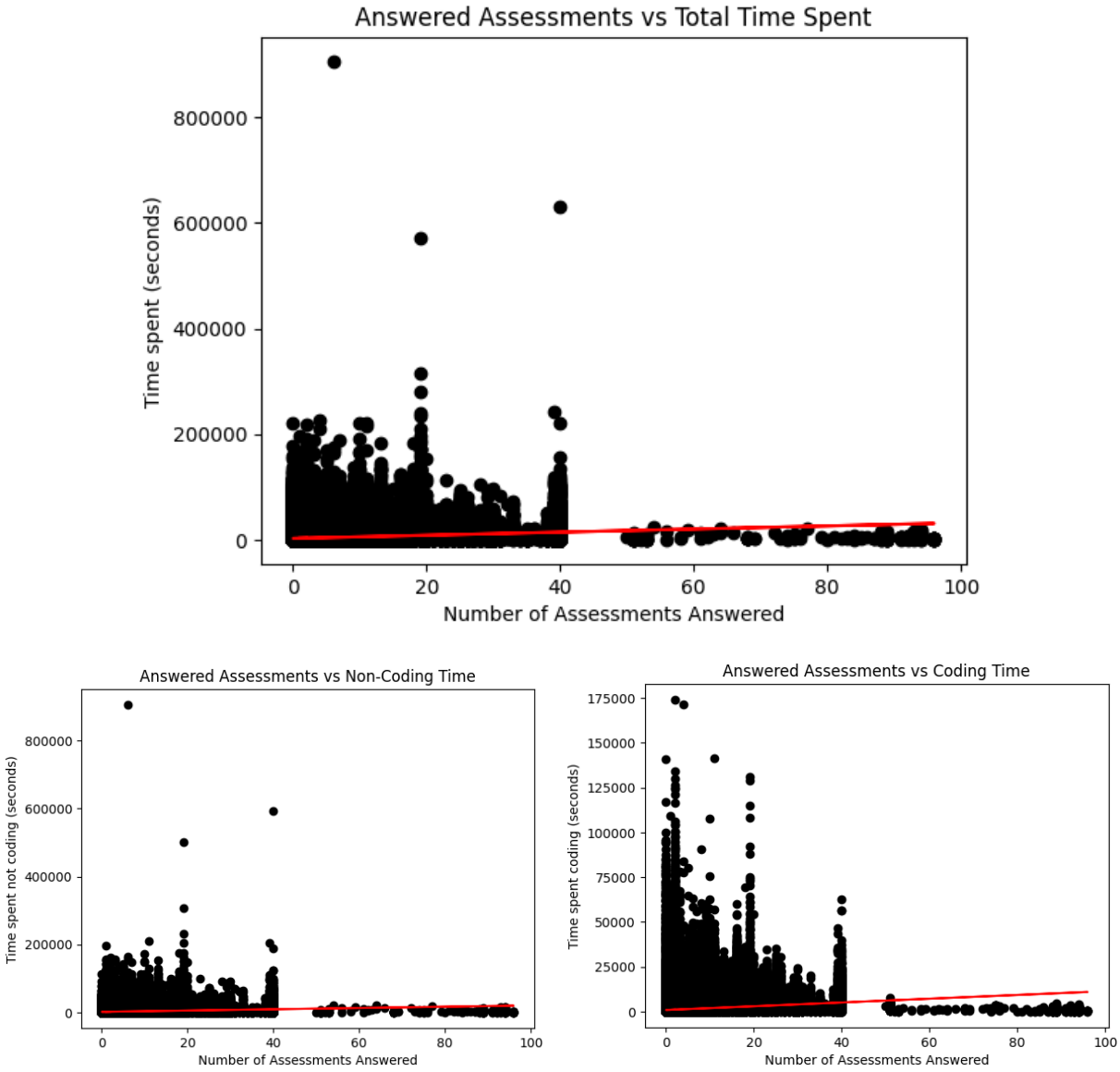

Figure 3: Answered Assessments correlated with types of Time Spent
Answered Assessments vs Total time spentRegression line: y = 294.207x + 2623.251
Answered Assessments vs Non-Coding Time Regression line: y = 189.427x + 1733.584
Answered Assessments vs Coding Time Regression line: y = 104.780x + 889.667

In an attempt to better understand the interplay between these fields, a multiple regression analysis was run, which showed statistical significance:

$$\text{grade} = -0.0006 \times \text{words} + 0.0006 * \text{coding\_time} + -3.84 \times 10^{-5} \times \text{non-coding\_time} + -0.196 \times \text{answeered\_assessments} + 84.252$$

While the R squared value is uninspiring at 0.005, notably all coefficients aside from coding time are negative.

Digging in to just how students are spending their time, a multiple regression again showed statistical significance with a similar (0.004) R squared value:

$$\text{grade} = 0.0005*\text{coding\_time} + -4.13 \times 10^{-6} \times \text{non-coding\_time} + 83.167$$

Again, we see coding time as having a positive and meaningful larger coefficient than non-coding time.

Conclusion

Our linear regression results seemingly indicated that what type of time students spent during assignments was unimportant. Notably reading and assessments seem just as much a part of coding as non-coding activities. When multiple regression analysis is run we begin to see the same results as we would expect from the literature – that students actively constructing code is more influential then non-coding time.

Given that time spent actively coding does not appear to be at the expense of other learning activities such as reading and assessments, even without deep context, we see the promise of structuring asynchronous learning experiences around opportunities for students to be actively engaging with code.

References

[1] L. E. Margulieux, and R. Catrambone, “Improving problem solving with subgoal labels in expository text and worked examples”, *Learning and Instruction* 42, pp. 58-71, 2016.

[2] C. T. Amelink, G. Scales, and J. G. Tront, "Student use and perception of textbooks in engineering education," *Journal of Engineering Education, vol. 110, no. 3*, pp. 585-601, 2021.

[3] M. Prensky, "Digital natives, digital immigrants part 1," *On the Horizon, vol. 9, no. 5*, pp. 1-6, 2001.

[4] D.H. Smith IV, Q. Hao, C. D. Hundhausen, F. Jagodzinski, J. Myers-Dean, and K. Jaeger, "Towards modeling student engagement with interactive computing textbooks: An empirical study," in Proceedings of the 52nd ACM Technical Symposium on Computer Science Education, 2021, pp. 914-920.

[5] A. T. Bates, G. Poole, and T. Bates, Effective teaching with technology in higher education: Foundations for success. Jossey-Bass, 2017.

[6] J. L. Jensen, T. A. Kummer, and P. D. d. M. Godoy, "Improvements from a flipped classroom may simply be the fruits of active learning," *CBE—Life Sciences Education, vol. 14, no. 1, p. ar5*, 2018.

[7] B. B. Morrison, L. E. Margulieux, B. J. Ericson, and M. Guzdial, "Subgoals help students solve Parsons problems," *Learning and Instruction, vol. 34*, pp. 63-71, 2015.

[8] T. Naps, G. Rößling, V. Almstrum, W. Dann, R. Fleischer, C. Hundhausen, A. Korhonen, L. Malmi, M. McNally, S. Rodger, & J. Á. Velázquez-Iturbide, "Exploring the role of visualization and engagement in computer science education," SIGCSE Bulletin, vol. 35, no. 2, pp. 131-152, 2002.

[9] H. G. Sigarchian, S. Logghe, R. Verborgh, W. de Neve, F. Salliau, and E. Mannens, "Hybrid e-TextBooks as comprehensive interactive learning environments," *Interactive Learning Environments*, 2018.

[10] R. Freedman and C. Adam, "Bridging the gap between academia and industry in computer science education," *ACM Inroads, vol. 7, no. 4*, pp. 48-51, 2016.

[11] C. Gordon, R. Lysecky, and F. Vahid, "The shift from static college textbooks to customizable content: A case study at zyBooks," in *2021 IEEE Frontiers in Education Conference (FIE)*, 2021, IEEE Press.

[12] T. Greer, Q. Hao, M. Jing, and B. Barnes, "On the effects of active learning environments in computing education," in *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*, 2019, pp. 267-272.

[13] B. Simon, C. Hundhausen, C. McDowell, L. Werner, H. Hu, & C. Kussmaul, "Students as teachers and communicators," in *The Cambridge Handbook of Computing Education Research*, 2019, pp. 827-857.

[14] L. A. Williams and R. R. Kessler, "All I really need to know about pair programming I learned in kindergarten," *Communications of the ACM, vol. 43, no. 5*, pp. 108-114, 2000.

[15] A. Edgcomb, F. Vahid, and R. Lysecky, "Students learn more with less text that covers the same core topics," in *2015 IEEE Frontiers in Education Conference (FIE)*, 2015, pp. 1-5.
[16] A. Edgcomb, F. Vahid, R. Lysecky, A. Knoesen, R. Amirtharajah, and M. L. Dorf, "Student performance improvement using interactive textbooks: A three-university cross-semester analysis," in the *Proceedings of the ASEE Annual Meeting*, 2015.
[17] J. Nielsen, "How Users Read on the Web," 1997. [Online]. Available: https://www.nngroup.com/articles/how-users-read-on-the-web/
[18] A. Edgcomb and F. Vahid, "Effectiveness of online textbooks vs. interactive web-native content," in *2014 ASEE annual conference*, 2014.
[19] C. Alvarado, B. B. Morrison, B. J. Ericson, M. Guzdial, B. Miller, & D. L. Ranum, "Performance and use evaluation of an electronic book for introductory Python programming," 2012.
[20] B. J. Ericson, L. E. Margulieux, and J. Rick, "Solving parsons problems versus fixing and writing code," in *Proceedings of the 17th Koli Calling Conference on Computing Education Research*, 2017, pp. 20-29.
[21] B. J. Ericson, M. J. Guzdial, and B. B. Morrison, "Analysis of interactive features designed to enhance learning in an ebook," in *Proceedings of the eleventh annual International Conference on International Computing Education Research*, 2015, pp. 169-178.
[22] F. P. Brooks, "The mythical man-month," *Datamation, vol. 20, no. 12*, pp. 44-52, 1974.
[23] E. Kula, E. Greuter, A. Van Deursen, and G. Georgios, "Factors Affecting On-Time Delivery in Large-Scale Agile Software Development," *IEEE Transactions on Software Engineering*, 2021.
[24] C. A. Shaffer and A. M. Kazerouni, "The Impact of Programming Project Milestones on Procrastination, Project Outcomes, and Course Outcomes: A Quasi-Experimental Study in a Third-Year Data Structures Course," in *Proceedings of the 52nd ACM Technical Symposium on Computer Science Education*, 2021, pp. 907-913.
[25] J. M. Allen, K. Downey, K. Miller, A. D. Edgcomb, & F. Vahid, "Many Small Programs in CS1: Usage Analysis from Multiple Universities," in *2019 ASEE Annual Conference & Exposition*, 2019.