On the Differences in Time That Students Take to Write Solutions to Programming Problems

Fabian Fagerholm
Department of Computer Science
Aalto University
Espoo, Finland
fabian.fagerholm@aalto.fi

Arto Hellas
Department of Computer Science
Aalto University
Espoo, Finland
arto.hellas@aalto.fi

***Abstract*—*Full research paper*—In this work, we study productivity differences in an introductory programming course. Focusing on a set of students who completed all programming assignments in the course, we quantify differences in productivity, measured through the time spent on completing the assignments. We focus both on the overall time needed to complete all programming assignments in the course, as well as on time spent on individual programming assignments. In addition, the effect of previous programming experience and difficulty of the programming assignment is considered. Our results show significant productivity differences between students. In addition, while programming experience influences productivity, a proportion of students who have never programmed before are faster in completing the programming assignments than students with considerable amounts of previous programming experience. Our results suggest that the classic credit-based or lecture hour based workload estimates of a course fit poorly to the whole course population in programming, suggesting that programming courses and training should be adjusted based on the participant.**

***Index Terms*—CS1, programming, productivity, time on task, novice, expert**

I. Introduction

Productivity differences between software engineers have been reported and discussed in various works [1]–[8]. Some quote differences in the orders of magnitude [1], while others suggest smaller differences [9]. The productivity of a software developer may vary based on the task or other factors [2], indicating that productivity is not a stable construct. Each developer has, however, started their career as a novice, often attending an introductory programming course.

In this work, we seek to quantify the productivity differences of students participating in an open introductory programming course. We are interested in determining whether the differences in productivity are similar to those reported in the context of software development, or whether the differences from the literature do not apply to those attending a course intended for novices. Information on the differences in productivity could be used, for example, for course design decisions.

Large productivity differences might stem from various factors, including prior exposure to programming; prior acquisition of beneficial personal work routines, cognitive strategies, or mental models of computing; and from personal characteristics, including aptitude for analytic tasks and personality traits that support working in a precise manner with sometimes tedious tasks [10]. Getting a picture of the productivity of individual students as well as groups of students could support teachers in personalising programming courses and is a step towards understanding student performance.

Productivity in real-life software engineering is impacted by a multitude of factors, and overall performance is ultimately judged in terms of organisational goals, such as financial profitability [11]. Still, individual productivity plays an important role in commercial software development: faster completion of tasks translates into cost savings, schedule benefits, and earlier delivery. If quality is simultaneously maintained or improved, there are further benefits for both the supplier and the customer. Thus understanding productivity is an important component in software engineering education, and means of improving and studying software engineering students' productivity are called for.

The analysis presented in this article focuses on students who completed all programming assignments in an open online programming course (n=173). As the students were working on the programming assignments, process data including keystrokes with timestamps were gathered for analysis purposes. Using this data, we quantify productivity and study it through multiple aspects.

This article is organized as follows. Next, in Section II, we outline related work on differences in programming productivity. In Section III, we discuss the context, data, research questions, and the research approach. Section IV outlines the results of our study, which are further discussed in Section V. Section VI concludes the work, outlining future research directions.

II. Background

One of the first studies that compared developer productivity was published by Sackman et al. in 1968 [12]. In the study, the authors compared the productivity of developers in programming and debugging tasks. The article has been later cited as pointing out that the productivity differences between developers can be as much as 28:1. The work has also been critiqued, however, as these highest productivity differences come from a setting where the developers had different tools [2], [9]. For example, Prechelt [9] points out that "*The oft-cited ratio of 28:1 for slowest to fastest work time*

in the experiment is plain wrong. The correct value is 14:1", continuing with data from their experiments, suggesting that "*we can say that typical work time differences between slower and faster programmers are more on the order of 2:1*". These findings have been echoed by others as well [13].

High throughput in completing programming or debugging tasks is just one criterion of being a good software engineer [14]. The tendency to produce high volumes of unmaintainable code, which might work in a specific situation, can lead to a code base that is difficult to maintain and later cause problems due to accumulating technical debt. A developer, in the end, does many other tasks in addition to programming as a part of their work; for example, in a study by Begel and Simon [15], new software developers spent a considerable amount of time on non-direct programming activities (communication, documentation, tools, project management, specifications, etc) in addition to the programming activities (working on bugs, programming, testing).

Experience in programming reduces development time [16] and influences productivity in programming related tasks. For example, experts are better at debugging source code, possibly because they are better at understanding the programs [17] and recognizing relevant information [18]. There are also differences between the actions of experts and non-experts – for example, non-experts are more likely to introduce new bugs to programs as they seek to fix existing problems [17]. On the other hand, there are studies that suggest that after a while, productivity does not increase. Lawrence, for example, points out that while they found an increase in productivity between developers with 0-1 years and 2-3 years of experience, no productivity increase was found between groups that had 2-3 years of experience and 3 or more years of experience [19].

Terminology in the area can be challenging, and the term productivity has been used various ways [19]. In educational contexts, performance is often viewed through the lens of performance in a course [20], while in software development, time that developers spend in developing or debugging software is often used [9], [12]. While the time is often recorded through observations or self-reports, the way in which time on a particular task is estimated can also be elaborated. For example, Takada et al. [21] studied developers' efficiency by monitoring their keystrokes. The keystrokes were aggregated and studied to identify actions on modifying code, which then led to a metric of developer performance quantifying developers' ability to fix issues in a system [21].

The use of keystroke data in analysing programmers has become more popular in the recent years [22]. For example, Leinonen et al. [23] noted that in the context of an introductory programming course, previous programming experience can be inferred to some extent from keystrokes: those who have programmed before tend to type programming-related character pairs faster than those who have not programmed previously. Timestamp data that is often included in keystroke data can also be used to estimate other factors in performance in introductory programming courses [24].

III. Methodology

A. Context and Data

The study was conducted using data from an open online programming course in Java[1], offered for free for anyone by the University of Helsinki in early 2019. The duration of the course is seven weeks, and the course covers the basics of programming in Java, starting from procedural programming (input/output, variables, conditionals, loops, methods), continuing with the use of elementary data structures in programming (lists, hashmaps), learning to create classes and objects, including objects that contain other objects, as well as lists, and learning basic sorting and searching algorithms.

The course is taken by both degree and non-affiliated students at the University of Helsinki. When starting the course, students fill in a research consent form and a background questionnaire outlining their previous programming experience. Answering the background questionnaire is voluntary and those who answer it are included in a raffle of movie tickets, even if they in the end choose that their data can not be used for research.

The workload of the course is 5 ECTS credits, which corresponds to approximately 125-150 hours of study. The course has an interactive online textbook accompanied with a set of weekly programming assignments (ca 15-40 assignments each week), which are automatically assessed. Students work on the course assignments individually using a modified version of Test My Code [25], a programming environment that provides feedback on the programming process, automatically assesses submissions, and records programming process data. When students work on programming assignments, key-level data of their programming process is collected. For each key press that changes source code in an assignment, an event outlining the change is stored. The event contains a diff entry detailing the change, a timestamp, the name of the assignment, and the identifier of the student.

Finally, whenever a student completes an assignment, they are prompted for a subjective measurement of difficulty of the assignment using a five-step scale from very easy to very difficult. Answering the question is voluntary, and the data is used to create an aggregate outline of the difficulty of the programming assignments in the course.

B. Research Questions and Approach

Our research questions for this study are as follows:

RQ1 What are the productivity differences between students in a programming course?

RQ2 How does previous programming experience influence students' productivity?

RQ3 How does the difficulty of a programming assignment influence students' productivity?

RQ4 What types of assignments show the highest differences in students' productivity?

RQ5 Is productivity consistent across the programming assignments?

[1] Version in English at: https://java-programming.mooc.fi/

TABLE I
MINIMUM, MAXIMUM, AVERAGE, STANDARD DEVIATION AND MEDIAN OF COMPLETED PROGRAMMING ASSIGNMENTS (OUT OF 176) AND PREVIOUS PROGRAMMING EXPERIENCE (IN HOURS).

	min	max	avg	stdev	median
Completed assignments	1	176	106	63	115
Programming experience	0	80000	599	3762	15

In this work, we define *productivity in an assignment* as the time that a student spends to complete the programming assignment, *productivity in all programming assignments* as the total time that a student spends in completing all the assignments, and *productivity difference* as the difference in productivity between two given students.

All productivity data is calculated using events stored by the programming environment. Pauses of more than five minutes are omitted in order to, heuristically, avoid including time that the student is disengaged from the programming task. We acknowledge that this likely also excludes time that the student spends studying the materials and the assignment handout, as well as time that the student uses for looking up information from external sources, and other similar activities.

In this analysis, we focus only on those students who completed all the programming assignments in the course. Moreover, to avoid bias from possible excessive collaboration and/or plagiarism, analysis of productivity differences focuses on populations (e.g. least productive student in the top quintile versus the most productive student in the bottom quintile) instead of comparing the most productive and the least productive individuals. Finally, when conducting statistical analyses, outliers have been excluded from the analysis (here, data points outside 3 standard deviations of the mean).

IV. RESULTS

A. Overview of Data

In total, 1972 students in the course provided research consent and responded to the questionnaire asking for their previous programming experience. The median completed programming assignments was 115 (out of 176), and the median self-reported previous programming experience in hours was 15. The statistics are summarized in Table I.

Out of the 1972 students, 173 students completed every assignment out of the 176 assignments (approx 9% of the included students). On average, students who completed all assignments spent approximately 32 hours on the programming assignments, when all pauses over 5 minutes are excluded.

B. Productivity Differences

Productivity differences were considered from two aspects: (1) the difference in total time that the students spent on completing all assignments, and (2) the difference in total time that the students spent on completing each individual assignment. Only those students who completed all course assignments were considered.

First, we studied the difference in total time spent on programming assignments. The least productive student in the top 10% spent 15 hours on the programming assignments, while the most productive student in the bottom 10% spent 51 hours on the programming assignments. Here, there is a 3.4-fold difference. Similarly, the least productive student in the top 20% spent 18 hours on the programming assignments in the course, while the most productive student in the bottom 20% spent 44 hours on the programming assignments. The difference is 2.4-fold.

Next, productivity differences in individual assignments were considered. When considering the least productive student of the top 10% and the most productive of the bottom 10%, the average productivity difference over the assignments was 12.3. Here, the smallest productivity difference in the assignments was 3.7, while the largest productivity difference was 136.0 – meaning that in one of the assignments, the slowest of the top 10% students completed the assignment approximately 136 times faster than the fastest of the bottom 10% students.

When considering the least productive student of the top 20% and the most productive of the bottom 20%, the average productivity difference over the assignments was 4.8. The smallest productivity difference in the assignments was 2.3, while the largest productivity difference was 20.4. Productivity differences over the assignments for the top 20% and bottom 20% are visualized in Figure 1.

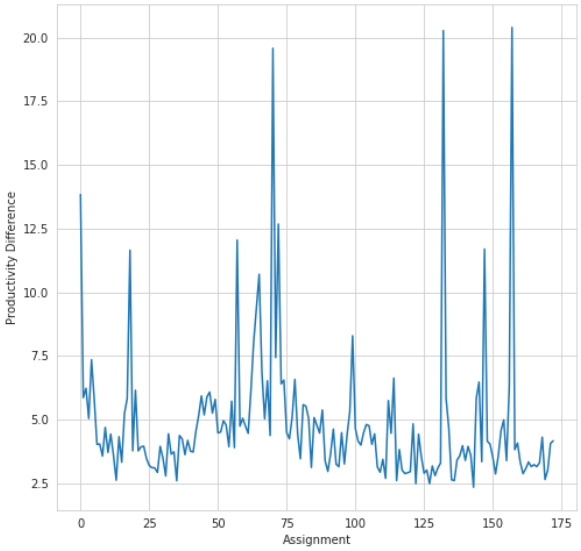

Fig. 1. Productivity differences across the course assignments for the fastest of the bottom 20% of the students and the slowest of the top 20% of the students. The x-axis indicates the assignment number, while the y-axis shows the productivity difference. The spikes in the chart predominantly highlight short assignments that allow large differences in productivity.

C. Productivity and Previous Experience

First, we studied the correlation between the previous programming experience in hours and the time that it takes to complete all the programming assignments. Using Pearson's r, a statistically significant but weak negative correlation was identified ($r = -0.23, p = 0.003$), implying that students with more programming experience are, on average, marginally faster in completing the programming assignments.

Then, we created two groups of students based on their previous programming experience, including only those who completed all programming assignments with no previous programming experience (n=17) and with over 1000 hours of previous programming experience (n=46). On average, the students who had no previous programming experience completed the assignments in 38 hours (stdev=15.4), while the students who had programmed previously completed the programming assignments in 27 hours (stdev=13.7).

Shapiro-Wilk test was used to test for normality of productivity in all assignments. The data was not normally distributed ($W = 0.87, p \approx 0$). Then, Kruskal-Wallis test was used to test whether the groups differ from each others. There is a statistically significant difference in the productivity in all assignments between the groups ($stat = 8.1, p = 0.004$) in favor of the students with previous programming experience.

Subsequently, we studied the difference in productivity between the groups. When considering the least productive student of the top 20% and the most productive of the bottom 20% with no previous experience, the average productivity difference was 1.6-fold. Similarly, when considering the least productive student of the top 20% and the most productive of the bottom 20% with previous experience, the average productivity difference was 2.5-fold.

When considering the slowest 20% of the students with previous programming experience and the fastest 20% of the students with no previous programming experience, the productivity difference is 1.3-fold in favor of the student with no experience. Similarly, when considering the slowest 10% of the students with previous programming experience and the fastest 10% of the students with no previous programming experience, the productivity difference is 2.3-fold in favor of the student with no experience.

The difference in the productivity of the populations is shown in the density plot in Figure 2. The line in the density plot is smoothed for visualization purposes. Students with previous programming experience are, on average, more productive when completing the programming assignments. At the same time, there is an overlap in the populations, indicating that some of the students with no previous programming experience complete assignments faster than some of the students with previous programming experience.

D. Productivity and Programming Assignment Difficulty

First, we studied whether the time spent on an assignment is related to the difficulty of the assignment. Using student-provided aggregate feedback on programming assignment difficulty and averaged time needed to complete an assignment,

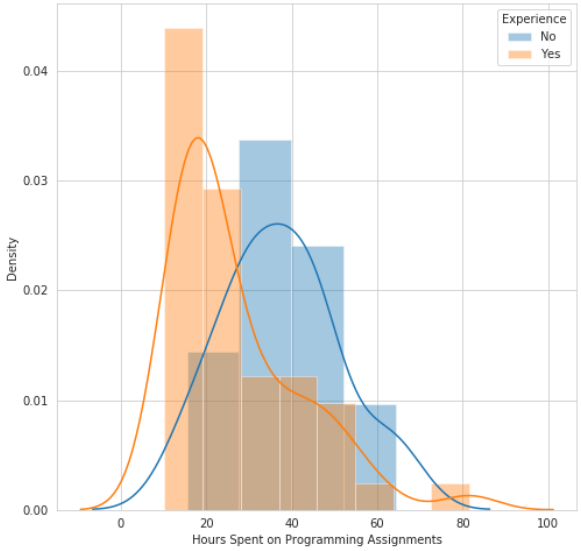

Fig. 2. Density plot showing the number of hours that students spent on completing all the programming assignments in the course. The plot shows two groups of students: students with no previous programming experience (blue), and students with 1000 hours of previous programming experience or more (orange).

we calculated Pearson's r to study the correlation between the variables. A strong statistically significant correlation was observed between the averaged programming assignment difficulty and the averaged time needed to complete the assignment ($r = 0.82, p \approx 0$). The correlation is visualized in Figure 3 using first order linear regression.

Next, we studied whether the productivity differences in assignments are related to the difficulty in assignments. Using student-provided aggregate feedback on programming assignment difficulty and the averaged productivity differences for individual assignments discussed in Section IV-B, we calculated Pearson's r to study the correlation. A statistically significant but weak correlation was identified ($r = -0.20, p = 0.009$), suggesting that the difficulty of an assignment may have a negligible effect on the productivity difference in the assignment. The data is visualized in Figure 4 using first order linear regression.

E. Programming Assignments and Productivity Differences

We then studied the ten assignments with the largest productivity differences and the ten assignments with the smallest productivity differences. The average productivity difference for the ten assignments with the largest productivity difference was 14.2, while the average productivity difference for the ten assignments with the smallest productivity difference was 2.6.

In general, the assignments with the highest productivity differences were such where the students were (1) learning specifics of data structures for the first time (e.g. learning

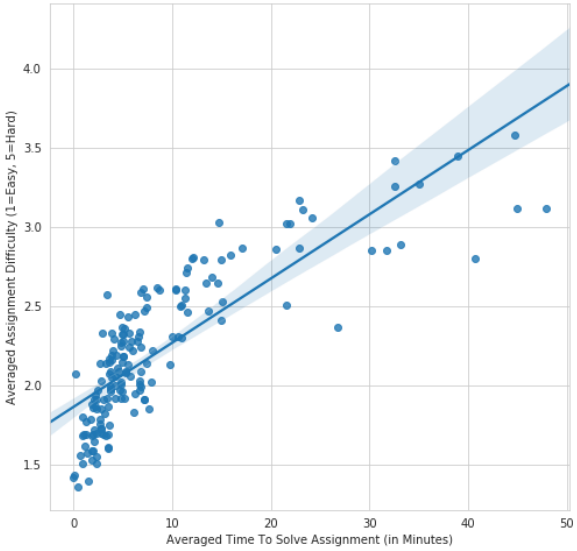

Fig. 3. Average difficulty of an assignment plotted against the average time to solve the assignment. The difficulty of an assignment is represented using a scale from 1 to 5, where smaller values indicate easier assignments, while the average time to solve an assignment is depicted in minutes.

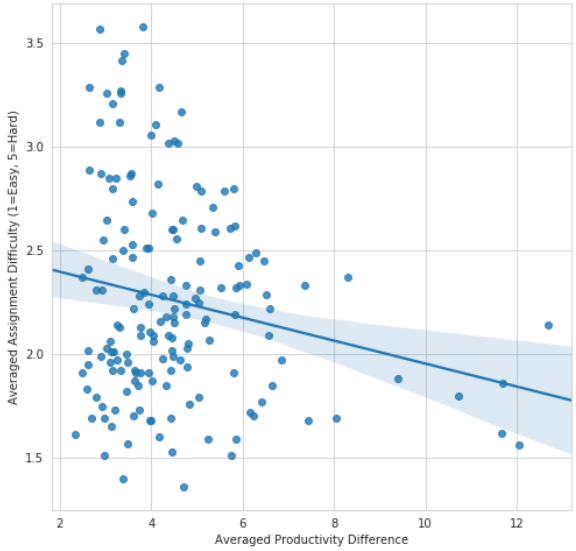

Fig. 4. Average difficulty of an assignment plotted against the productivity difference of an assignment. The difficulty of an assignment is represented using a scale from 1 to 5, where smaller values indicate easier assignments. Productivity difference is calculated as the ratio of the productivity of the slowest student in the fastest 20% of students and the fastest student in the slowest 20% of students.

about indexing and retrieving a value from the end of a list); (2) combining mathematics and programming concepts (e.g. creating a good hash function for dates); (3) creating programs that showed particular programming-related effects (e.g. writing a program that throws the NullPointerException with a particular input, writing a program that throws the IndexOutOfBoundsException with a particular input); and (4) writing small programs with degrees of freedom (i.e. students had more room to choose what they wanted to implement).

The assignments with the lowest productivity differences were (1) very well specified; (2) asking the student to implement a program that is very similar to an example given in the course material; (3) asking the student to implement a *variant* of a program that the student had previously implemented; (4) larger in size but still well-specified.

F. Consistency of Productivity over Programming Assignments

Finally, we studied whether students' productivity is consistent across the assignments. This was conducted through calculating students' rank in each programming assignment, and then studying the average ranks and their standard deviation. First, we calculated Pearson's r to study the correlation between the variables. A moderate statistically significant correlation was observed between the averaged rank of a student and the standard deviation of the rank ($r = 0.50, p \approx 0$). The data is visualized in Figure 5 using first order linear regression.

This suggests that, on average, the higher the students' rank is, the more the rank is scattered over the assignments. In other words, the data does not support that all students would be consistently ranked.

V. Discussion

A. Productivity Differences

Our analysis focused on a small subset of students who had completed all the programming assignments in the course (9% of the available population). When considering the overall time taken to complete the course assignments, there were over 3-fold differences between the slowest student from the top 10% and the fastest student in the bottom 10%. It is likely that if the analysis would have focused on the top 5% and bottom 5%, the differences would have been larger.

When looking at the productivity differences within programming assignments, the largest productivity difference within a single assignment (when looking at the slowest student from the top 10% and the fastest student from the bottom 10%) was over 136-fold. Here, however, the difference was observed in a very small assignment on indexes and lists, where the fastest students could complete it within a handful of seconds. While such differences may be seen in e.g. debugging velocity of experts and novices, where finding a simple bug may take significant amounts of time for a novice [26], we acknowledge that such assignments should not be used for quantifying overall productivity differences. Instead, they could, assuming that the assignments are small, have potential to be used as a rapid test of comprehension of the current topic, for example.

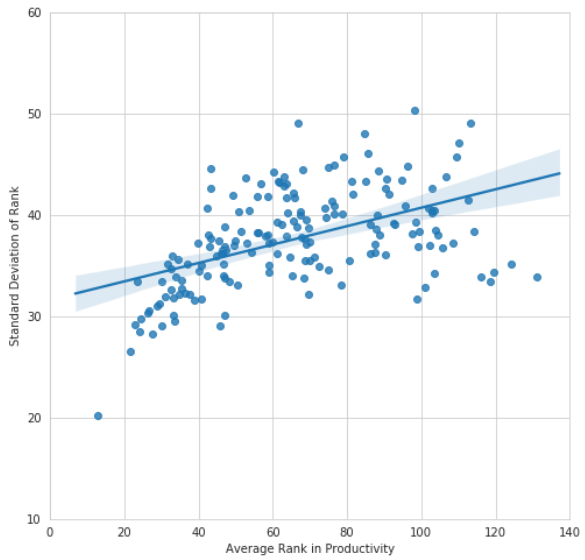

Fig. 5. Average rank of a student plotted against the standard deviation of the rank of the student, calculated based on productivity in each assignment.

While it is possible that some of the productivity differences could be due to unwanted behavior, i.e. excessive collaboration, we sought to remove the influence of such behavior from the analysis by focusing on subsets of students. Consequently, in addition to not studying the absolute fastest students, we also chose not to study the absolute slowest students; this was conducted to avoid including off-task behavior into the analysis.

While we focused only on the students who completed all programming assignments, it is possible that a student who did not complete all of the assignments actually spent more time on the course, as the programming assignments in the course often build on top of the previous assignments. On the other hand, it is possible that some students may have skipped assignments due to some of them being repetitive, and there may have been students who would be faster in completing the assignments. In the current analysis, we sought to quantify productivity differences, and chose to not quantify how practice within the course, i.e. completed or skipped programming assignments, influences productivity – this is left for future work.

B. Novices and Non-Novices

In the course, a majority of the participants had 15 hours or less previous programming experience. When considering the overall effect of previous programming experience on productivity, we observed a statistically significant but weak correlation between the variables. It is possible that due to the skew of the data, the effect could in fact be larger than what the observed correlation ($r^2 = 0.05$) suggests.

To determine actual productivity differences between novices and non-novices, we further divided the students into two populations. One population had reported that they have no previous programming experience (n=17), while the other population had reported that they had programmed at least for one thousand hours (n=46). Here, there was a significant difference in terms of productivity between the populations, where the students with previous programming experience completed the assignments noticeably faster.

At the same time, there was an overlap in the populations in terms of productivity. Some of the students who had stated that they have no previous programming experience completed all the course assignments faster than some of the students with previous programming experience. This was contrary to our intuition that any student who has programmed for at least one thousand hours should complete programming assignments in an introductory programming course faster than those who have not programmed before. At this point, we do not know what student-specific factors contributed to this result.

We must note that the student population who had plenty of programming experience was over-represented: students with no previous programming experience were less likely to complete all the programming assignments. Regardless, this result suggests that some novices either start the course at a level where they can complete programming assignments faster than others, or that the differences grow over time. Consequently, this also suggests that when entering industry and taking on their first programming jobs, some of the novices may perform faster – after a while – than some of the developers who have already been working at the industry for a short duration.

C. Quality and Productivity

Our analysis only focused on assignments that the students had completed, and we did not discern differences e.g. in terms of quality including readability and maintainability. While the course materials encourage writing programs that can be understood by others, it is possible that some of the students sacrificed quality and readability of their programs in favor of completing the programs faster. That is, we do not claim that high productivity in our study would imply high quality.

Similarly, when considering learning, it is possible that students who spent more time on the programming assignments have learned more during the process than those who spent less time on the assignments. Being faster at completing programming assignments does not automatically imply understanding the content better – on the contrary, it may even be that students who fly through the assignments do not consider the reasons why something is implemented the way it is instructed. Similarly, a mistake that takes time to fix may yield a better learning outcome than not doing the mistake at all.

D. Productivity and Programming Courses

Even though our analysis did not consider the time that students spend on reading course materials or the time that students spend on non-programming activities such as respond-

ing to questionnaires and essays, the observed productivity differences indicate that some may complete the course faster than others. When considering students who reported that they have no previous programming background, there was a 1.6-fold productivity difference between the top 20% and the bottom 20%. As the analysis focused only on students who had completed all the assignments, it is possible that the actual productivity differences are larger.

If a programming course is supposed to take a specified amount of time from a student, such as the estimated 5 ECTS workload, which translates to 125-150 hours of work, the contents of the course should be adjusted based on students' performance. This is, however, not practically feasible, as the course is a pre-requisite for subsequent courses. One way to account for our results in the design of a programming course would be to make sure that the programming course would have a clear minimum set of skills that the target population could reach in the specified time-frame, while additional activities and learning opportunities could be provided to more advanced students.

E. Productivity and Programming Assignments

When considering individual programming assignments, we observed over 100-fold productivity differences between the top 10% and bottom 10% students. Such large differences in productivity can be, however, explained by the programming assignments and students' previous programming experience; tasks that are trivial to those with previous programming experience can take a long while from novices, especially if they get stuck to e.g. a syntactical construct.

While the 100-fold productivity difference was an extreme example, the differences in productivity within assignments were larger than the differences in productivity over the course. This was supported also through our analysis of students' consistency in the programming assignments, where we observed that students' productivity rank (i.e. the rank in productivity within an assignment) varies considerably; this may suggest that different students struggle with different concepts.

Moreover, when looking into the difficulty of the programming assignments, we observed that difficulty and the time it takes to complete the task are interlinked. At the same time, the average difficulty of an assignment seems to have little connection with the productivity differences between students, which further suggests that not all students struggle with all assignments.

F. Limitations of Work

Our study comes with a range of limitations which we discuss next. First, as the analysis focused on a small part of students in the course, we cannot claim that the results would be similar for the whole course population, or in other courses. Second, it is possible that some of the students did not complete all assignments themselves, but they may have received help from others; that is, it is possible, that the productivity reported here does not reflect the actual productivity of the participants. Third, in the analysis, we used a five-minute window as an indicator of possible off-task behavior; here, hypothetical students who continued their work after a pause of four minutes and 59 seconds were significantly penalized in terms of productivity when compared to those, who continued their work after a pause of five minutes and one second. Overall, the issue of measuring time from logs has been discussed e.g. in [27]. We sought to address the second and the third issue, which are both related to the way in which time is estimated, by considering only subpopulations of students, i.e. the slowest student in the top quintile and the fastest student in the bottom quintile. Fourth, when studying programming background, we focused on the number of hours as a metric of previous experience – we acknowledge that the metric is not ideal [28]. It is, however, likely that previous experience with a non-related programming language could be less beneficial than previous experience with the language the course used. Moreover, it is possible that different students understood the term previous programming experience differently; some may have included e.g. high school courses using spreadsheets or writing HTML, while others may not. Finally, due to analyzing only subpopulations of students, we did not discern extreme differences in productivity – it is highly likely that there are even larger differences in productivity, e.g. when studying the top 1% and the bottom 1%.

VI. Conclusions

In this work, we studied productivity differences, measured as time spent, between students in a programming course using data collected from students' programming process. Our analysis revealed noticeable productivity differences between students – our answer to the first research question *"What are the productivity differences between students in a programming course?"*, is: The productivity differences between the top quintile and bottom quintile over the course is at least two-fold, while the productivity differences within individual assignments can be larger.

In the course, students were surveyed about their previous programming experience in terms of hours programmed. We studied the connection between the self-reported previous programming experience and students' productivity. Our answer to the second research question *"How does previous programming experience influence students' productivity?"* is: There was a weak negative correlation between hours programmed and productivity, implying that those with previous programming experience were more productive. Moreover, when considering students with no previous programming experience and students who had previously programmed at least 1000 hours, the differences in productivity were significant. At the same time, some of the students who had not previously programmed were more productive than some of the students who had programmed at least 1000 hours.

After completing an assignment, students could answer a question on the difficulty of the assignment. To determine whether the difficulty of an assignment influences productivity, we studied the connection between the average difficulty of an assignment and average productivity in the assignment, as

well as differences in productivity. Our answer to the third research question *"How does the difficulty of a programming assignment influence students' productivity?"* is: There is a strong correlation between the difficulty of the assignment and the time it takes to solve the assignment. At the same time, there is a weak negative correlation between the average assignment difficulty and the average productivity difference, suggesting that the difficulty of an assignment has only little explanatory power over the productivity differences in an assignment.

We also studied the assignments with the smallest and largest productivity differences. Our answer to the fourth research question *"What types of assignments show the highest differences in students' productivity?"* is: The largest productivity differences can be explained by (small) assignment size and skill differences, while the smallest productivity differences were observed in well specified assignments similar to assignments that students had already worked on.

Finally, we studied the consistency of productivity within the programming assignments by analyzing whether students' are ranked the same across the programming assignments (in terms of productivity). Our answer to the fifth research question *"Is productivity consistent across the programming assignments?"* is: No. While the most productive students tend to remain among the most productive students, there are considerable productivity differences within the student population across the course assignments.

Performance differences between students are important to consider in curriculum development. In previous literature, previous programming experience and consequently performance has been connected to, e.g., career motivations [29] and course retention [30]. Courses with wide differences in performance and knowledge may, on the other hand, demotivate weaker students if they compare themselves against more experienced peers. Our results support the notion of providing an accelerated track for more experienced students in introductory programming courses [31]. This would leave more teaching resources for novices and could ease integration into the learning community.

As a part of our future work, instead of solely focusing on students who completed all the assignments, we are looking into the larger student population in order to determine to what extent the productivity differences observed in our study generalize to the overall population, as well as to determine to what extent skipping individual assignments influences productivity in the course. We are also looking into factors that could explain some of the productivity, including the structure and quality of students' programs, with the goal of identifying practices that could lead to higher productivity.

REFERENCES

[1] H. Sackman, "Man-computer communication: Experimental investigation of user effectiveness," in *Proceedings of the Sixth SIGCPR Conference on Computer Personnel Research*, ser. SIGCPR '68. New York, NY, USA: ACM, 1968, pp. 93–105. [Online]. Available: http://doi.acm.org/10.1145/1142648.1142658

[2] B. Curtis, "Substantiating programmer variability," *Proceedings of the IEEE*, vol. 69, no. 7, pp. 846–846, July 1981.

[3] H. Mills, *Software Productivity*. New York, NY, USA: Dorset House, 1983.

[4] T. DeMarco and T. Lister, *Peopleware: Productive Projects and Teams*. New York, NY, USA: Dorset House, 1999.

[5] D. N. Card, "A software technology evaluation program," *Information and Software Technology*, vol. 29, no. 6, pp. 291–300, 1987.

[6] B. W. Boehm and P. N. Papaccio, "Understanding and controlling software costs," *IEEE Transactions on Software Engineering*, vol. 14, no. 10, pp. 1462–1477, Oct 1988.

[7] J. D. Valett and F. E. McGarry, "A summary of software measurement experiences in the software engineering laboratory," *Journal of Systems and Software*, vol. 9, no. 2, pp. 137–148, 1989.

[8] B. W. Boehm, C. Abts, A. W. Brown, S. Chulani, B. K. Clark, E. Horowitz, R. Madachy, D. J. Reifer, and B. Steece, "Software cost estimation with cocomo ii," 2000.

[9] L. Prechelt, "The 28: 1 grant-sackman legend is misleading, or: How large is interpersonal variation really," Tech. Rep., 1999.

[10] B. Curtis, "Fifteen years of psychology in software engineering: Individual differences and cognitive science," in *Proceedings of the 7th International Conference on Software Engineering*, ser. ICSE '84. Piscataway, NJ, USA: IEEE Press, 1984, pp. 97–106. [Online]. Available: http://dl.acm.org/citation.cfm?id=800054.801956

[11] C. Sadowski and T. Zimmerman, Eds., *Rethinking Productivity in Software Engineering*. Apress, 2019.

[12] H. Sackman, W. J. Erikson, and E. E. Grant, "Exploratory experimental studies comparing online and offline programming performance," *Commun. ACM*, vol. 11, no. 1, pp. 3–11, Jan. 1968. [Online]. Available: http://doi.acm.org/10.1145/362851.362858

[13] W. R. Nichols, "The end to the myth of individual programmer productivity," *IEEE Software*, vol. 36, no. 5, pp. 71–75, Sep. 2019.

[14] P. L. Li, A. J. Ko, and J. Zhu, "What makes a great software engineer?" in *Proceedings of the 37th International Conference on Software Engineering - Volume 1*, ser. ICSE '15. Piscataway, NJ, USA: IEEE Press, 2015, pp. 700–710. [Online]. Available: http://dl.acm.org/citation.cfm?id=2818754.2818839

[15] A. Begel and B. Simon, "Novice software developers, all over again," in *Proceedings of the fourth international workshop on computing education research*. ACM, 2008, pp. 3–14.

[16] E. Chrysler, "Some basic determinants of computer programming productivity," *Commun. ACM*, vol. 21, no. 6, pp. 472–483, Jun. 1978. [Online]. Available: http://doi.acm.org/10.1145/359511.359523

[17] L. Gugerty and G. Olson, "Debugging by skilled and novice programmers," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ser. CHI '86. New York, NY, USA: ACM, 1986, pp. 171–174. [Online]. Available: http://doi.acm.org/10.1145/22627.22367

[18] M. P. Robillard, W. Coelho, and G. C. Murphy, "How effective developers investigate source code: An exploratory study," *IEEE Trans. Softw. Eng.*, vol. 30, no. 12, pp. 889–903, Dec. 2004. [Online]. Available: https://doi.org/10.1109/TSE.2004.101

[19] M. J. Lawrence, "Programming methodology, organizational environment, and programming productivity," *J. Syst. Softw.*, vol. 2, no. 3, pp. 257–269, Sep. 1981. [Online]. Available: http://dx.doi.org/10.1016/0164-1212(81)90023-6

[20] Z. Karimi, A. Baraani-Dastjerdi, N. Ghasem-Aghaee, and S. Wagner, "Links between the personalities, styles and performance in computer programming," *J. Syst. Softw.*, vol. 111, no. C, pp. 228–241, Jan. 2016. [Online]. Available: http://dx.doi.org/10.1016/j.jss.2015.09.011

[21] Y. Takada, K. Matsumoto, and K. Torii, "A programmer performance measure based on programmer state transitions in testing and debugging process," in *Proceedings of the 16th International Conference on Software Engineering*, ser. ICSE '94. Los Alamitos, CA, USA: IEEE Computer Society Press, 1994, pp. 123–132. [Online]. Available: http://dl.acm.org/citation.cfm?id=257734.257752

[22] P. Ihantola, A. Vihavainen, A. Ahadi, M. Butler, J. Börstler, S. H. Edwards, E. Isohanni, A. Korhonen, A. Petersen, K. Rivers *et al.*, "Educational data mining and learning analytics in programming: Literature review and case studies," in *Proceedings of the 2015 ITiCSE on Working Group Reports*. ACM, 2015, pp. 41–63.

[23] J. Leinonen, K. Longi, A. Klami, and A. Vihavainen, "Automatic inference of programming performance and experience from typing patterns," in *Proceedings of the 47th ACM Technical Symposium on Computing Science Education*, ser. SIGCSE '16. New York, NY, USA: ACM, 2016, pp. 132–137. [Online]. Available: http://doi.acm.org/10.1145/2839509.2844612

[24] J. Leinonen, L. Leppänen, P. Ihantola, and A. Hellas, "Comparison of time metrics in programming," in *Proceedings of the 2017 ACM Conference on International Computing Education Research*, ser. ICER '17. New York, NY, USA: ACM, 2017, pp. 200–208. [Online]. Available: http://doi.acm.org/10.1145/3105726.3106181

[25] A. Vihavainen, T. Vikberg, M. Luukkainen, and M. Pärtel, "Scaffolding students' learning using test my code," in *Proceedings of the 18th ACM conference on Innovation and technology in computer science education*, 2013, pp. 117–122.

[26] M. C. Jadud, "Methods and tools for exploring novice compilation behaviour," in *Proceedings of the second international workshop on Computing education research*, 2006, pp. 73–84.

[27] J. Leinonen, L. Leppänen, P. Ihantola, and A. Hellas, "Comparison of time metrics in programming," in *Proceedings of the 2017 acm conference on international computing education research*, 2017, pp. 200–208.

[28] R. S. Duran, J.-M. Rybicki, A. Hellas, and S. Suoranta, "Towards a common instrument for measuring prior programming knowledge," in *Proceedings of the 2019 ACM Conference on Innovation and Technology in Computer Science Education*, ser. ITiCSE '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 443–449. [Online]. Available: https://doi.org/10.1145/3304221.3319755

[29] A. Master, S. Cheryan, A. Moscatelli, and A. N. Meltzoff, "Programming experience promotes higher stem motivation among first-grade girls," *Journal of experimental child psychology*, vol. 160, pp. 92–106, 2017.

[30] A. Hellas, P. Ihantola, A. Petersen, V. V. Ajanovski, M. Gutica, T. Hynninen, A. Knutas, J. Leinonen, C. Messom, and S. N. Liao, "Predicting academic performance: a systematic literature review," in *Proceedings Companion of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education*, 2018, pp. 175–199.

[31] A. Fisher and J. Margolis, "Unlocking the clubhouse: the carnegie mellon experience," *ACM SIGCSE Bulletin*, vol. 34, no. 2, pp. 79–83, 2002.