Aalto University

OPEN ACCESS

This is an electronic reprint of the original article.
This reprint may differ from the original in pagination and typographic detail.

Edwards, John; Leinonen, Juho; Hellas, Arto

A study of keystroke data in two contexts: written language and programming language influence predictability of learning outcomes

Published in:
SIGCSE 2020 - Proceedings of the 51st ACM Technical Symposium on Computer Science Education

DOI:
10.1145/3328778.3366863

Published: 26/02/2020

Document Version
Peer-reviewed accepted author manuscript, also known as Final accepted manuscript or Post-print

Please cite the original version:
Edwards, J., Leinonen, J., & Hellas, A. (2020). A study of keystroke data in two contexts: written language and programming language influence predictability of learning outcomes. In *SIGCSE 2020 - Proceedings of the 51st ACM Technical Symposium on Computer Science Education* (pp. 413-419). (Annual Conference on Innovation and Technology in Computer Science Education). ACM. https://doi.org/10.1145/3328778.3366863

This material is protected by copyright and other intellectual property rights, and duplication or sale of all or part of any of the repository collections is not permitted, except that material may be duplicated by you for your research use or educational purposes in electronic or print form. You must obtain permission for any other use. Electronic or print copies may not be offered, whether for sale or otherwise to anyone who is not an authorised user.

A Study of Keystroke Data in Two Contexts: Written Language and Programming Language Influence Predictability of Learning Outcomes

John Edwards
john.edwards@usu.edu
Utah State University
Logan, Utah

Juho Leinonen
juho.leinonen@helsinki.fi
University of Helsinki
Helsinki, Finland

Arto Hellas
arto.hellas@aalto.fi
Aalto University
Espoo, Finland

ABSTRACT

We study programming process data from two introductory programming courses. Between the course contexts, the programming languages differ, the teaching approaches differ, and the spoken languages differ. In both courses, students' keystroke data – timestamps and the pressed keys – are recorded as students work on programming assignments. We study how the keystroke data differs between the contexts, and whether research on predicting course outcomes using keystroke latencies generalizes to other contexts. Our results show that there are differences between the contexts in terms of frequently used keys, which can be partially explained by the differences between the spoken languages and the programming languages. Further, our results suggest that programming process data that can be collected non-intrusive in-situ can be used for predicting course outcomes in multiple contexts. The predictive power, however, varies between contexts possibly because the frequently used keys differ between programming languages and spoken languages. Thus, context-specific fine-tuning of predictive models may be needed.

CCS CONCEPTS

• **Social and professional topics** → *Computing education*; *CS1*; • **Computing methodologies** → *Supervised learning by classification.*

KEYWORDS

keystroke analysis, digraphs, keystroke data, programming process data, predicting performance, educational data mining

ACM Reference Format:
John Edwards, Juho Leinonen, and Arto Hellas. 2020. A Study of Keystroke Data in Two Contexts: Written Language and Programming Language Influence Predictability of Learning Outcomes. In *The 51st ACM Technical Symposium on Computer Science Education (SIGCSE '20), March 11–14, 2020, Portland, OR, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3328778.3366863

SIGCSE '20, March 11–14, 2020, Portland, OR, USA
© 2020 Copyright held by the owner/author(s). Publication rights licensed to ACM. This is the author's version of the work. It is posted here for your personal use. Not for redistribution. The definitive Version of Record was published in *The 51st ACM Technical Symposium on Computer Science Education (SIGCSE '20), March 11–14, 2020, Portland, OR, USA*, https://doi.org/10.1145/3328778.3366863.

1 INTRODUCTION

Learning and teaching happens in various contexts across the world. Classrooms with chalkboards may be the primary context associated with learning in some places, while in some other locations, learning activities may primarily be organized in small groups or through apprenticeships without classrooms. The way teaching is organized influences learning outcomes [12]. The situation is similar in learning and teaching programming: programming is taught all around the world, with numerous spoken languages, in numerous programming languages, with numerous teaching approaches, and with varying learning outcomes [4, 18, 31].

In addition to learning outcomes, contextual differences influence how research results transfer from one context to another [14]. A result from one context may be applicable to another context immediately, but it can also be the case that an approach has limited value in another context [27]. This could be due to the data and the context where the data is produced, the methodology, or even the application of the methodology or used libraries [14].

Recently, analysis of student keystrokes has received attention within the computing education research community [19]. Research has suggested that students can be accurately identified based solely on their keystrokes [23]. Similarly, keystroke data has been used to distinguish high and low performing students [21, 30]: such work has mainly focused on Java as the programming language.

Using keystroke data from two programming courses with different teaching languages, programming languages, and teaching approaches, we study *to what extent do keystroke data generalize across contexts.* Methodologically, we partially replicate the work by Thomas et al. [30] and Leinonen et al. [21], further extending the work by a qualitative analysis of the keystroke data.

This study is motivated by the problem of replicability. In general, replication studies in CS education are scarce [11]. While there is a need for more studies that seek to replicate research findings in other contexts [14], there are also issues with how replication studies are valued and consequently preferred [1]. Evidence of the (lack of) generalizability of research may allow the community as a whole to identify tacit factors that influence research outcomes.

This article is organized as follows. In Section 2, we briefly review work on predicting programming performance, keystroke analytics, and their intersection. This is followed by the methodology, including a description of the contexts, data, and research questions. In Section 4, we outline the results, which are further discussed in Section 5. Finally, in Section 6, we conclude the article and outline possible directions for future work.

2 RELATED WORK

2.1 Predicting Programming Performance

There exists a vast body of research into predicting academic performance [13]. Such studies range from identifying factors that contribute towards course outcomes to developing and improving methodologies that are used for predicting course outcomes. Information on students at risk of possible drop-out is however rarely used for pedagogical interventions [13]. This could be partially due to the workload required to conduct such studies – in the past, many of such studies have been based on information collected through e.g. surveys or through other means that require effort [13].

Recently, predicting students' performance using data collected from the programming process has increased in popularity. For example, Jadud et al. [15] proposed an approach that uses the occurrence of compilation errors in successive compilation events in predicting course outcomes. This work has been extended by Watson et al. [35], who added temporal information on how fast students fix possible compilation errors. Other streams of work on the topic include using more data from the programming process including steps needed to solve programming problems as well as the correctness of the problem [2] and information on runtime errors [5]. Students' study behavior has been analyzed also outside of the programming environment – for example, Porter et al. have used clicker data from Peer Instruction courses to identify struggling students [29].

The granularity of the data influences the information available within the data [32]. Data from programming environments are collected using multiple granularities: some may only collect students' submissions, while others may collect every keystroke [14]. This also influences the predictive models that one can build – constructing e.g. models that use individual key-presses [21] is not possible when only submissions are available.

2.2 Keystroke Analysis

Keystrokes combined with their timings have been analyzed for purposes ranging from identifying the individual typing [10, 16, 17, 28], recognizing the emotions of the typist [9], and to inferring demographic factors of the person typing [3].

Most keystroke analysis approaches rely on building a typing profile of the person [17]. In the case of identifying someone based on typing, the typing profile of someone writing is compared to existing typing profiles to determine the identity of the typist. The analysis is usually based on how fast a person types specific character pairs, or digraphs [6, 10, 23]. A digraph is a pair of adjacent characters. For example, the word *good* has three digraphs: *go*, *oo*, and *od*. A typing profile usually contains the average digraph latencies for a person, i.e. how fast on average the person types different character pairs.

The context of typing can affect identification accuracies. Villani et al. [34] had people write on both laptop and desktop computers and found that when the same keyboard was used, identification accuracies were relatively high (> 90%), but when people typing on desktop were being identified based on laptop typing profiles and vice versa, the accuracy decreased to around 60%. Leinonen et al. [20] found that identifying students in a programming exam based on typing profiles from programming assignments is harder than identifying students within a single context, i.e. during programming assignments. In a similar fashion, Peltola et al. [26] studied how the type of text being written affects identification. They built typing profiles of students during programming assignments and examined how well students can be identified in a programming exam and when writing essays in natural language. Their results show that identification accuracies decrease when the type of text is changed between building the typing profiles and identification, in their case, when identifying students writing an essay in natural language based on typing profiles built from programming keystrokes. Thus, when predicting how students will perform based on keystrokes, the context might affect the results.

2.3 Programming Performance and Keystrokes

Thomas et al. [30] examined using keystroke timings to infer students' programming performance. They divided digraphs into categories based on the characters of the digraph. They found that specific digraph types correlate moderately with students' exam scores. The categories with the most predictive power were numeric digraphs (both characters are numbers), digraphs where one character is a browsing character (e.g. arrow keys), and "edge" digraphs where both characters are from different categories.

Leinonen et al. [21] partially replicated Thomas et al.'s experiment and extended it by also studying to what extent students' prior programming experience could be automatically inferred based on their typing. Similar to Thomas et al., they found that numeric digraphs and "edge" digraphs had the best predictive power. Additionally, they found that both students' exam performance and students' previous programming experience can be inferred partly based on their typing – better performing students and students with more previous programming experience were faster at typing specific digraphs that are related to programming (e.g. *i+* and digraphs containing special characters such as *{}* and *//*).

3 METHODOLOGY

3.1 Context and Data

3.1.1 University A. Python keystroke data was collected at a mid-sized public university in the US. Data was collected during the first eight weeks of a CS1 course using a custom web-based IDE. In the ninth week students were transitioned to a mainstream Python IDE without keystroke logging capability. 265 students participated in the study. There were three sections, two taught by one instructor and the remaining section taught by a different instructor. In this paper we report results from five programming assignments, with each assignment consisting of two projects, for a total of 10 programming projects. Each pair of projects was assigned at the same time and was due at the same time, but the two assigned tasks are unrelated. In general, one project is a mathematical calculation with text-based output, and the other project is a turtle graphics-based drawing. In the programming environment the development window has two tabs, one for each project. The student can work on either project at any time. The projects are manually assessed.

The students at University A complete the programming assignments reported in this paper using an online Python programming environment called Phanon that logs programming events including

keystrokes, pastes, switches between programming tasks (by clicking on a different task tab), and run attempts. Each event is logged with a timestamp. This paper reports results only on keystrokes.

The outcome variable is student score on the midterm exam, given ten weeks into the semester, or two weeks after the last of the keystroke data is collected. The exam is computer-based and includes multiple choice, true/false, and simple fill in the blank questions. The fill in the blank questions require short answers, e.g. "what function outputs text to the screen?" and "what arguments should be passed to the given function to achieve the following output?".

3.1.2 University B. Java keystroke data was collected in a 7-week CS1 course at a public research-first university in Finland, which is a Northern European country. The teaching, including materials and assignments, are given in Finnish. The course population is rather homogeneous and practically everyone has done their primary and secondary education in Finland. The course in question had one weekly lecture and walk-in labs, where students were guided by the course teacher and a pool of course assistants. A total of 303 students participated in the study.

In the course, students work on tens of programming assignments each week. The assignments are interleaved in the course material. Whenever a new topic is learned, a handful of programming assignments are worked on to internalize the topic. Several of the assignments, when combined, build into larger programs. Programming assignments are completed using NetBeans, which is a desktop IDE, accompanied with the `Test My Code` [33] plugin that collects the keystroke data, downloads course assignments for students, and provides the capabilities for running, testing, and submitting the assignments. The assignments are automatically assessed.

The outcome variable is student score on the final exam given at the end of the 7-week course. The exam is computer-based and includes programming assignments similar to the ones that students have worked on during the course.

The contexts are summarized in Table 1.

Table 1: Summary of contexts

Variable	University A	University B
Instruction	Lectures w/sections	Lecture & labs
Language (prog.)	Python	Java
Language (inst.)	English	Finnish
Participants	265	303
Environment	Web-based	Desktop
Projects	10	133
Assessment	Manual	Automatic
Exam	Midterm	Final
Exam content	MCQ, Fill in the blank	Programming

3.2 Research Questions and Approach

Our research questions for this study are as follows.

RQ1 How do the contexts differ in terms of collected keystroke data?

RQ2 To what extent do digraph latencies predict course outcomes in the studied contexts?

To answer RQ1, we analyze the distributions of keystrokes both quantitatively and qualitatively, focusing on the differences between the contexts.

To answer RQ2, we first study correlation of students' average digraph latencies (lower value means faster typing) and the outcome variable. This is followed by the analysis outlined in [21]; we identify the most frequent digraphs (i.e. character-pairs) and their average latencies, and then evaluate a Random Forest classifier[1] for predicting course outcomes. As the outcome variable for predicting performance, we use median split similar to previous studies on predicting performance [2, 21]. This means that for each student, we predict whether they will be in the top or the bottom half of the exam population performance-wise. After the median split, the group sizes for Java are 150 and 153 (majority classifier accuracy 50.5%), while for Python the group sizes are 130 and 135 (majority classifier accuracy 50.9%). In addition to reporting classifier accuracies, we report Matthew's Correlation Coefficients (MCCs) of the classifiers. The analysis is performed using the Scikit-learn [25] machine learning library for Python.

We only consider students' digraph latencies that are between 10 and 750 milliseconds similar to previous work [21]. Further, as the full data has over 5000 different digraphs for Python and over 8000 digraphs for Java, feature selection is used to prune down the features to avoid overfitting. The SelectKBest feature selection method (with ANOVA F-values for scoring features) from Scikit-learn [25] was used to select the top 50 features with the most predictive power.

To avoid reporting overly positive (or pessimistic) results caused by a random seed, which influences the outcomes of the predictive model, we ran evaluations several times using different random seeds. For the MCCs, data was split into a randomly selected train and a test set (0.9 / 0.1 split) one hundred times, and for the Random Forest, we used a tenfold cross-validation and ran the classifier with ten random seeds. Results are averaged over the runs and we report the standard deviations in addition to accuracy and MCC.

4 RESULTS

4.1 Differences Between Contexts

Our first research question is how keystroke data differs between the two contexts. Students at University A had written on average 11879 digraphs (std = 7345) while students at University B had written on average 38366 digraphs (std = 23441). There was no difference between the average typing speed of students between the contexts: students writing Java programs had an average digraph latency of 221 milliseconds (std = 29) and students writing Python programs had an average digraph latency of 218 milliseconds (std = 28).

While the raw typing speed was very similar, the distribution of digraphs differs between the contexts. Figure 1 shows the distributions of the most common 16 digraphs (excluding "`delete` →

[1]The following parameters were used: n_estimators=200, criterion="entropy", max_features="log2", max_depth=10.

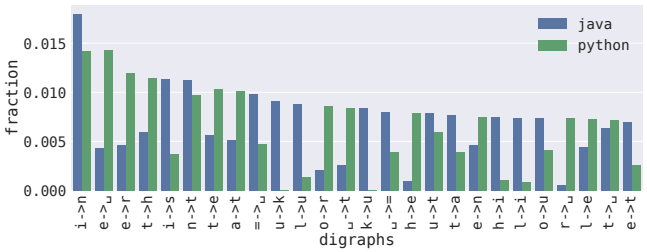

Figure 1: Digraph distributions. The y-axis is the fraction of total keystroke pairs a given digraph accounts for across all students and assignments in the course. The "delete → delete" digraph is more frequent by an order of magnitude than any other digraph (at 0.21 and 0.18 for Java and Python, respectively) and is omitted from the chart for clarity. The "␣" character represents a space.

delete", which was the most common digraph in both contexts by a large margin). Differences between spoken language (e.g. u → k being more common in Finnish than in English), programming-language (e.g. o → r being more common in Python than in Java), and instructional approach / guidelines to code quality (e.g. = → ␣ i.e. space, being more common at University B than at University A) were observed. These differences are further discussed in Section 5.1.

4.2 Keystroke Latencies and Course Outcomes

Our second research question is to what extent do digraph latencies predict course outcomes in the studied contexts. We look at the problem in two ways: first, we look for possible correlation between average typing speed (measured through average digraph latencies) and course outcomes, and then evaluate the applicability of a Random Forest classifier for the task.

As the digraph latencies are not normally distributed, we used Spearman rank correlation for the first analysis. Looking at the Spearman rank correlation between average typing speed and performance in the exam, we found that in the Python course, students' typing speed, as measured by average latencies, had a statistically significant but very weak correlation $r = -0.20$ ($p = 0.001$) with the exam score. In the Java course, no statistically significant correlation was observed $r = -0.05$ ($p = 0.44$).

Next, we constructed Random Forest classifiers for both contexts. We first considered predicting the course outcomes of all students, assigning students who dropped out from the course a 0 from the exam. When students were divided into two categories based on the median exam score, the accuracy of the Random Forest classifier for the Java course was 68%, while the accuracy of the classifier for the Python course was 62%. The MCC score for Java was 0.39 and 0.29 for Python, indicating weak to very weak correlation.

Subsequently, we considered predicting the course outcomes for only those students who attended the course exam. In the Java course, 256 students out of 303 participants attended the exam, while in the Python course, 263 out of the 265 participants attended the exam. Here, the accuracy of the Random Forest classifier was 72% for the Java course, while the accuracy of the classifier for the Python course did not change (62%). The MCC score for Java was 0.40 and 0.25 for Python, indicating weak to very weak correlation. The results of the predictive modeling are summarized in Table 2.

Finally, we analyzed the top 50 digraphs that provide the most value for the classifier (listed in Table 3). These digraphs further illustrate the difference between the programming languages: noticeably fewer special character combinations and numeric combinations are used in the Python context than in the Java context.

5 DISCUSSION

Our first research question that addresses the differences in keystroke data in the different contexts is discussed Sections 5.1 and 5.3. Our second research question that addresses to what extent digraph latencies work as a predictor of course outcomes, and whether the work generalizes to different contexts, is discussed in Sections 5.2 and 5.3.

5.1 Contextual Differences in Digraphs

Studying the most common digraphs (shown in Figure 1), we identified three main differences in digraphs between the contexts. The results discussed in this section have no obvious link to the difference in explanatory power between the contexts, but they may help clarify why digraphs can predict outcomes in the first place.

Native spoken language. The first contextual difference is the native spoken language of the students. In the Java course the students were primarily native Finnish speakers, and the materials and assignments were given in Finnish. This is reflected in the digraph distribution by the large difference in frequency in the digraphs "k → u" and "u → k". Both of these digraphs are common in Finnish but relatively uncommon in English. This difference is explained by the fact that the Finnish word for "number" is "luku", a word used frequently in input prompts, output, and comments.

Programming language. The second contextual difference is related to programming language. The digraph distribution shows a large difference in the frequency of the digraph "o → r" which is four times more common in the Python context than the Java context. This is partially due to Python using the keyword or for boolean disjunction, whereas Java uses the || characters (which are nearly non-existent in Python). Another example (not shown in Figure 1) is "r → ␣" which is 15 times more common in Python than Java. This could be because the Java keyword for is often followed by the (character and Python's for is followed by a space.

Instruction. The third contextual difference is that of instruction in the course. For example, in the Java course the instructor encouraged students to pad both sides of the = assignment operator with whitespace while the Python instructors did not. This is reflected in the "= → ␣" and "␣ → =" digraphs, which are far more prevalent in the Java course than the Python course. Similarly, the above mentioned difference between the "r → ␣", which we attributed to the programming language, could be also due to practices – had the instructors in the Java context enforced students to use a space after writing for, the difference could have been more subtle.

	Java (all students)	Java (exam attendees)	Python (all students)	Python (exam attendees)
MCC (std)	0.39 (0.17)	0.40 (0.19)	0.29 (0.18)	0.25 (0.18)
Accuracy (std)	68% (7%)	72% (6%)	62% (9%)	62% (9%)

Table 2: Average Matthew's Correlation Coefficients (MCCs) and 10-fold cross-validation scores, and their standard deviations in the parentheses. For the MCCs, 100 runs of Random Forest with different random seeds were ran, and for the 10-fold cross-validation 10 runs with different seeds were run. Scores were calculated separately for all course attendees and only those students who attended the exam.

Python

★ ␣→=	M→A	a→g	h→s	p→u
★ ␣→\	N→E	a→w	l→u	r→s
"→"	N→O	b→l	m→s	t→s
"→h	O→N	★ ,→-	n→c	u→a
'→t	O→R	d→d	n→f	u→b
1→2	R→E	del→f	n→n	u→g
2→.	S→e	e→h	n→o	u→i
9→0	T→I	f→u	o→␣	w→n
F→O	U→s	g→n	o→n	x→p
I→O	W→o	g→u	p→i	y→s

Java

␣→c	2→5	★ <→S	f→i	s→␣
★ !→=	3→2	S→E	f→o	s→d
★ &→ret	4→0	U→E	★ g→>	t→␣
'→s	5→0	★ a→>	g→a	★ t→<
1→.	5→9	d→u	j→i	★ t→>
1→0	8→0	d→y	l→n	t→V
1→8	9→1	del→9	ret→i	★ u→>
1→9	★ :→:	del→x	o→f	x→u
2→3	★ <→K	del→y	★ p→<	★ {→del
2→4	★ <→L	e→q	★ r→>	★ |→del

Table 3: Top 50 digraph features with the most predictive power in each of the two contexts (excluding delete → delete). Digraphs involving special characters are starred. `del` is the delete or backspace key, `ret` is the return, or newline, key, and ␣ is the spacebar.

5.2 Contextual differences in predicting outcomes

In this section we discuss contextual differences that may account for the difference in explanatory power between the Java course and the Python course.

Course design. The keystroke data from the Java course has more explanatory power than that of the Python course. One possible explanation is that the better accuracy is simply because the data from the Java course is roughly three times the size of the Python data. This explanation is consistent with the results of Leinonen et al. [21], where prediction accuracy increased as the teaching term progressed and the amount of training data increased. Another explanation that is discussed more in the following section is that exams in the Java course assess typing skill by requiring the student to type code whereas there is very little typing of code in the Python course's exam. An additional difference in the courses is that more students dropped out of the Java course before the first exam. This may have lead to selection bias, with students who dropped out possibly being more difficult to classify. A final difference that we note is that the IDE used in the Java course includes autocompletion whereas the Python IDE does not. It may be that code that can be autocompleted occurs less frequently in the data, given that students learn how to use autocompletion.

Difference in programming language. The Java data results in a higher ratio of digraphs with special characters among high-performing features than in the Python context (see Table 3) which we discuss further in the following subsection. An additional difference is that many of the top-performing Java digraph features are numerical digraphs (both characters are numbers). This result is similar to the findings by Leinonen et al. [21] and Thomas et al. [30], both of which used Java as the programming language (Thomas et al. also used Ada in addition to Java). Curiously, the effect is less visible in Python, where only 2 of the top 50 digraphs are numeric while Java has 12 such features. This difference would seem attributable to the difference in programming language, but it is also possible that it is a factor of course design, i.e., if the Java course had more mathematical assignments then it may be that the numeric digraphs have more power in the Java course simply because they are more common.

5.3 Insights into digraphs as predictive features

Our results support the hypothesis that digraphs can be used for predicting course outcomes. Our results also shed further light on why digraphs may have predictive power:

Conjecture: Some CS1 exams also test students' ability to type. This simple explanation is supported at least in part by our results. The outcome variable in the Java study is the score on the final exam which includes programming problems similar to those in the assignments. Thus, in addition to testing a student's ability to analyze a problem and design a solution, the exam is also testing the student's typing ability. The Python exam requires very little typing and is not predicted as well by digraph features as the Java exams, supporting this explanation.

Conjecture: There exist underlying cognitive structures common to problem solving and typing characters that are uncommon in natural

language. Such structures would have important implications in understanding the process of writing programs. As seen in Table 3, 15 of the top 50 digraph features in the Java context involve special characters not commonly used in natural language, while the Python context has only 3 such digraphs. This is an important support for the idea that special characters are better predictors and explains why our Java data had better predictive power than the Python data, and additionally supports our conjecture regarding underlying cognition in both problem solving and typing unnatural text. This idea, however, may not be completely compatible with research suggesting that computer programming is composed of distinct skills with very different cognitive loads. Many researchers break programming into two skills: programming language syntax construction and problem solving [8, 22, 24] (Du Boulay [7] breaks problem solving into two additional skills called *structures* and *pragmatics*). Our results suggest that problem solving and construction of language syntax may be skills that are not mutually exclusive.

Conjecture: Students who are adept at typing are spending less time on the typing and instead using that time to become fluent in the other aspects of programming. This conjecture is not supported by our data. Define time spent typing as the sum of digraph latencies that are in the range $10 - 750$ milliseconds [6, 21] and time spent not typing as the sum of latencies in the range 751 milliseconds to 5 minutes (if greater then 5 minutes then we consider the student disengaged). With these definitions, the Python students spent only 18% of their total programming time typing. Even with a more generous typing cutoff latency of two seconds, the time spent typing is still only 30%. With raw typing time taking up such a small amount of total time we suggest that any small savings in typing would result in negligible improvement in programming. These breaks in programming should be further studied, however.

5.4 Limitations

We have shown results from two different contexts, but it is not certain whether the results would generalize to any other context. It is possible, for example, that digraphs collected from a Python course taught in Finnish could have more predictive over the course outcomes than the one in our study, which was taught in English.

We did not study to what extent the accuracy of the models change when using data solely, for example, from the first week of the courses. Thus, it remains an open question whether digraphs are useful for, for example, early detection of struggling students.

There could be other confounding variables that explain our results such as students' previous experience with computers. It is possible that the predictive power of digraphs is a result of one or more underlying variables, some of which may also be measured in other approaches that are used for predicting course outcomes. No demographic student data, including socioeconomic status, academic history, or physical/motor impairment, was obtained for this study, and as such, these could not be studied.

In the analysis, we split students into two categories – high and low performing students – with a median split. We acknowledge that this is not ideal as any two students near the median might end up in either of the two categories.

Finally, the interpretation of digraphs having predictive power over course outcomes are naturally subjective, as one could consider only near absolute predictive power as useful. At the same time, any improvement over a random guess baseline shows some value indicating that the digraphs could be studied further, and potentially also incorporated into other models.

6 CONCLUSIONS

In this work, we studied keystroke data from two introductory programming courses. The courses differ from each others in terms of pedagogy, spoken language, programming language, number of assignments, programming environment, and so on.

Our first research question, *how do the contexts differ in terms of collected keystroke data*, was answered through analysis of digraph distributions. We found that specific context differences, e.g., programming language, pedagogy, and native spoken language of the students were exhibited in the distributions. However, somewhat surprisingly, average typing speed was unaffected by context. We expected that typing speed in the Python context would be higher due to Python being closer to typing natural spoken language. Another surprising result was that typing speed in the Python context was correlated with exam score while the Java typing speed showed no correlation at all. As we do not have an explanation for this result, it may be a good candidate for future study.

Our second research question, *to what extent do digraph latencies predict course outcomes in the studied contexts*, yielded positive results. The explanatory power of the digraphs is small to moderate; when distinguishing students over or below the class median, the accuracy of the constructed Random Forest classifier was 62% for Python and 72% for Java. While the predictive power of digraphs is not large, our study showed that at least it is relatively consistent across two very different contexts.

In answering our second research question we discovered additional insights into possible reasons why keystroke data in general, and digraph latencies specifically, are predictive of course outcomes. We found fairly strong evidence to support the idea that exams may be unduly assessing student typing ability, evidence including the fact that the better-predicting Java context included exams that assessed students' abilities to write code in an IDE. Through analysis of the best-predicting digraph features we also gained insight into the possibility that typing text not common in natural language is linked to other aspects of programming ability. Despite these insights, we are unable to confidently identify the individual variables that could explain the differences in the predictive power due to the notable differences between the contexts. We second the call from [13], asking for researchers to explicitly outline their research contexts when reporting outcomes; only through detailed information on how the courses are conducted can we draw inferences on the factors that contribute to the outcomes.

As a part of our future work, we are looking into exploring the individual contextual factors further. We are interested in how the spoken language influences how students write programs and consequently how familiar students are with writing particular language-specific digraphs. Furthermore, we are looking into targeted practice of digraphs that most influence the predictive models, which will provide additional information on maturation: to what extent does learning to type particular digraphs faster influence the predictive power of the models.

REFERENCES

[1] Alireza Ahadi, Arto Hellas, Petri Ihantola, Ari Korhonen, and Andrew Petersen. 2016. Replication in computing education research: researcher attitudes and experiences. In *Proceedings of the 16th Koli Calling International Conference on Computing Education Research*. ACM, 2–11.

[2] Alireza Ahadi, Raymond Lister, Heikki Haapala, and Arto Vihavainen. 2015. Exploring machine learning methods to automatically identify students in need of assistance. In *Proceedings of the eleventh annual International Conference on International Computing Education Research*. ACM, 121–130.

[3] David Guy Brizan, Adam Goodkind, Patrick Koch, Kiran Balagani, Vir V Phoha, and Andrew Rosenberg. 2015. Utilizing linguistically enhanced keystroke dynamics to predict typist cognition and demographics. *International Journal of Human-Computer Studies* 82 (2015), 57–68.

[4] P Brusilovsky et al. 1994. Teaching Programming to Novices: A Review of Approaches and Tools. (1994).

[5] Adam S Carter, Christopher D Hundhausen, and Olusola Adesope. 2015. The normalized programming state model: Predicting student performance in computing courses based on programming behavior. In *Proceedings of the eleventh annual International Conference on International Computing Education Research*. ACM, 141–150.

[6] Paul S Dowland and Steven M Furnell. 2004. A long-term trial of keystroke profiling using digraph, trigraph and keyword latencies. In *IFIP International Information Security Conference*. Springer, 275–289.

[7] Benedict Du Boulay. 1986. Some difficulties of learning to program. *Journal of Educational Computing Research* 2, 1 (1986), 57–73.

[8] John M Edwards, Erika K Fulton, Jonathan D Holmes, Joseph L Valentin, David V Beard, and Kevin R Parker. 2018. Separation of syntax and problem solving in Introductory Computer Programming. In *2018 IEEE Frontiers in Education Conference (FIE)*. IEEE, 1–5.

[9] Clayton Epp, Michael Lippold, and Regan L Mandryk. 2011. Identifying emotional states using keystroke dynamics. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 715–724.

[10] R Stockton Gaines, William Lisowski, S James Press, and Norman Shapiro. 1980. *Authentication by keystroke timing: Some preliminary results.* Technical Report.

[11] Qiang Hao, David H. Smith IV, Naitra Iriumi, Michail Tsikerdekis, and Andrew J. Ko. 2019. A Systematic Investigation of Replications in Computing Education Research. *ACM Trans. Comput. Educ.* 19, 4, Article 42 (Aug. 2019), 18 pages. https://doi.org/10.1145/3345328

[12] John Hattie. 2008. *Visible learning: A synthesis of over 800 meta-analyses relating to achievement.* routledge.

[13] Arto Hellas, Petri Ihantola, Andrew Petersen, Vangel V Ajanovski, Mirela Gutica, Timo Hynninen, Antti Knutas, Juho Leinonen, Chris Messom, and Soohyun Nam Liao. 2018. Predicting academic performance: a systematic literature review. In *Proceedings Companion of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education*. ACM, 175–199.

[14] Petri Ihantola, Arto Vihavainen, Alireza Ahadi, Matthew Butler, Jürgen Börstler, Stephen H Edwards, Essi Isohanni, Ari Korhonen, Andrew Petersen, Kelly Rivers, et al. 2015. Educational data mining and learning analytics in programming: Literature review and case studies. In *Proceedings of the 2015 ITiCSE on Working Group Reports*. ACM, 41–63.

[15] Matthew C Jadud. 2006. Methods and tools for exploring novice compilation behaviour. In *Proceedings of the second international workshop on Computing education research*. ACM, 73–84.

[16] Rick Joyce and Gopal Gupta. 1990. Identity authentication based on keystroke latencies. *Commun. ACM* 33, 2 (1990), 168–176.

[17] M. Karnan, M. Akila, and N. Krishnaraj. 2011. Biometric personal authentication using keystroke dynamics: A review. *Applied Soft Computing* 11, 2 (2011), 1565 – 1573. https://doi.org/10.1016/j.asoc.2010.08.003 The Impact of Soft Computing for the Progress of Artificial Intelligence.

[18] Theodora Koulouri, Stanislao Lauria, and Robert D Macredie. 2015. Teaching introductory programming: A quantitative evaluation of different approaches. *ACM Transactions on Computing Education (TOCE)* 14, 4 (2015), 26.

[19] Juho Leinonen. 2019. *Keystroke Data in Programming Courses.* Ph.D. Dissertation. University of Helsinki.

[20] Juho Leinonen, Krista Longi, Arto Klami, Alireza Ahadi, and Arto Vihavainen. 2016. Typing patterns and authentication in practical programming exams. In *Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education*. ACM, 160–165.

[21] Juho Leinonen, Krista Longi, Arto Klami, and Arto Vihavainen. 2016. Automatic Inference of Programming Performance and Experience from Typing Patterns. In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education (SIGCSE '16)*. ACM, New York, NY, USA, 132–137. https://doi.org/10.1145/2839509.2844612

[22] Marcia C Linn. 1985. The cognitive consequences of programming instruction in classrooms. *Educational Researcher* 14, 5 (1985), 14–29.

[23] Krista Longi, Juho Leinonen, Henrik Nygren, Joni Salmi, Arto Klami, and Arto Vihavainen. 2015. Identification of programmers from typing patterns. In *Proceedings of the 15th Koli Calling Conference on Computing Education Research*. ACM, 60–67.

[24] Richard E Mayer, Jennifer L Dyck, and William Vilberg. 1986. Learning to program and learning to think: what's the connection? *Commun. ACM* 29, 7 (1986), 605–610.

[25] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 12 (2011), 2825–2830.

[26] Petrus Peltola, Vilma Kangas, Nea Pirttinen, Henrik Nygren, and Juho Leinonen. 2017. Identification based on typing patterns between programming and free text. In *Proceedings of the 17th Koli Calling International Conference on Computing Education Research*. ACM, 163–167.

[27] Andrew Petersen, Jaime Spacco, and Arto Vihavainen. 2015. An exploration of error quotient in multiple contexts. In *Proceedings of the 15th Koli Calling Conference on Computing Education Research*. ACM, 77–86.

[28] Paulo Henrique Pisani and Ana Carolina Lorena. 2013. A systematic review on keystroke dynamics. *Journal of the Brazilian Computer Society* 19, 4 (2013), 573.

[29] Leo Porter, Daniel Zingaro, and Raymond Lister. 2014. Predicting student success using fine grain clicker data. In *Proceedings of the tenth annual conference on International computing education research*. ACM, 51–58.

[30] Richard C Thomas, Amela Karahasanovic, and Gregor E Kennedy. 2005. An investigation into keystroke latency metrics as an indicator of programming performance. In *Proceedings of the 7th Australasian conference on Computing education-Volume 42*. Australian Computer Society, Inc., 127–134.

[31] Arto Vihavainen, Jonne Airaksinen, and Christopher Watson. 2014. A systematic review of approaches for teaching introductory programming and their influence on success. In *Proceedings of the tenth annual conference on International computing education research*. ACM, 19–26.

[32] Arto Vihavainen, Matti Luukkainen, and Petri Ihantola. 2014. Analysis of source code snapshot granularity levels. In *Proceedings of the 15th Annual Conference on Information technology education*. ACM, 21–26.

[33] Arto Vihavainen, Thomas Vikberg, Matti Luukkainen, and Martin Pärtel. 2013. Scaffolding students' learning using test my code. In *Proceedings of the 18th ACM conference on Innovation and technology in computer science education*. ACM, 117–122.

[34] Mary Villani, Charles Tappert, Giang Ngo, Justin Simone, H St Fort, and Sung-Hyuk Cha. 2006. Keystroke biometric recognition studies on long-text input under ideal and application-oriented conditions. In *2006 Conference on Computer Vision and Pattern Recognition Workshop (CVPRW'06)*. IEEE, 39–39.

[35] Christopher Watson, Frederick WB Li, and Jamie L Godwin. 2013. Predicting performance in an introductory programming course by logging and analyzing student programming behavior. In *2013 IEEE 13th International Conference on Advanced Learning Technologies*. IEEE, 319–323.