Keystroke Data in Programming Courses

Leinonen, Juho

Helsingin yliopisto

2019-12-20

Department of Computer Science
Series of Publications A
Report A-2019-8

Keystroke Data in Programming Courses

Juho Leinonen

Doctoral dissertation, to be presented for public discussion with the permission of the Faculty of Science of the University of Helsinki, in Room 167, Athena building, University of Helsinki, on the 20th of November, 2019 at 12 o'clock.

University of Helsinki
Finland

Supervisors
Arto Hellas, Petri Ihantola, Tommi Mikkonen,
Arto Klami and Petri Myllymäki
University of Helsinki, Finland

Pre-examiners
Judithe Sheard, Monash University, Australia
Mikko-Jussi Laakso, University of Turku, Finland

Opponent
Nickolas Falkner, University of Adelaide, Australia

Custos
Petri Myllymäki, University of Helsinki, Finland

Contact information

Department of Computer Science
P.O. Box 68 (Pietari Kalmin katu 5)
FI-00014 University of Helsinki
Finland

Email address: info@cs.helsinki.fi
URL: http://cs.helsinki.fi/
Telephone: +358 2941 911

Copyright © 2019 Juho Leinonen
ISSN 1238-8645
ISBN 978-951-51-5603-7 (paperback)
ISBN 978-951-51-5604-4 (PDF)
Computing Reviews (1998) Classification: K.3.1, K.3.2
Helsinki 2019
Unigrafia

Keystroke Data in Programming Courses

Juho Leinonen

Department of Computer Science
P.O. Box 68, FI-00014 University of Helsinki, Finland
juho.leinonen@helsinki.fi

PhD Thesis, Series of Publications A, Report A-2019-8
Helsinki, November 2019, 56+53 pages
ISSN 1238-8645
ISBN 978-951-51-5603-7 (paperback)
ISBN 978-951-51-5604-4 (PDF)

Abstract

Data collected from the learning process of students can be used to improve education in many ways. Such data can benefit multiple stakeholders of a programming course. Data about students' performance can be used to detect struggling students who can then be given additional support benefiting the student. If data shows that students have to read a certain section of the material multiple times, it could indicate either that that section is possibly more important than others, or it might be unclear and could be improved, which benefits the teacher. Data collected through surveys can yield insight into students' motivations for studying. Ultimately, data can increase our knowledge of how students learn benefiting educational researchers.

Different kinds of data can be collected in online courses. In programming courses, data is typically collected from tools that are specifically made for learning programming. These tools include Integrated Development Environments (IDEs), program visualization tools, automatic assessment tools, and online learning materials. The granularity of data collected from such tools varies. Fine-grained data is data that is collected frequently, while coarse-grained data is collected less frequently. In a programming course, coarse-grained data might include students' submissions to exercises, whereas fine-grained data might include students' actions within the IDE such as editing source code. An example of extremely fine-grained data is keystroke data, which typically includes each key pressed while typing together with a timestamp that tells when exactly the key was pressed.

In this work, we study what benefits there are to collecting keystroke data in programming courses. We explore different aspects of keystroke data that could be useful for research and to students and educators. This is studied by conducting multiple quantitative experiments where information about students' learning or the students themselves is inferred from keystroke data. Most of the experiments are based on examining how fast students are at typing specific character pairs.

The results of this thesis show that students can be uniquely identified solely based on their typing whilst they are programming. This information could be used in online courses to verify that the same student completes all the assignments. Excessive collaboration can also be detected automatically based on the processes students take to reach a solution. Additionally, students' programming experience and future performance in an exam can be inferred from typing, which could be used to detect struggling students. Inferring students' programming experience is possible even when data is made less accurate so that identifying individuals is no longer feasible.

Computing Reviews (1998) Categories and Subject Descriptors:
K.3.1 Computer Uses in Education
K.3.2 Computer and Information Science Education

General Terms:
Experimentation, Measurement

Additional Key Words and Phrases:
keystroke data, keystroke analysis, keystroke dynamics, programming data, programming process data, source code snapshots, biometrics, data privacy, data anonymization, replication, educational data mining

Acknowledgements

First of all, I would like to thank my supervisors Arto Hellas, Petri Ihantola, Tommi Mikkonen, Arto Klami and Petri Myllymäki. You all have provided me great guidance during different parts of my academic career. I especially thank Arto Hellas, who in 2015 gave me the opportunity to join the Agile Education Research Group solely based on my interest in research in this area when I had not yet even completed my bachelor's degree. You have been an amazing mentor and friend.

I would like to thank my pre-examiners Judy Sheard and Mikko-Jussi Laakso for their excellent and thoughtful feedback on the thesis. I also thank Nick Falkner for agreeing to be the opponent at my defence.

I am grateful for the Department of Computer Science, the University of Helsinki, and the Doctoral Programme in Computer Science (DoCS) for the possibility to conduct my research here. I greatly appreciate the Helsinki Doctoral Education Network in Information and Communications Technology (HICT) grant which funded this work. Additionally, I thank Pirjo Moen for answering all the questions I have had related to the process of getting a PhD.

During the last four years, first as a research assistant and then as a PhD student, I have had the opportunity to work on awesome things with awesome people. I thank all my co-authors, my past and present colleagues at the Agile Education Research Group and elsewhere, especially Henrik Nygren, Matti Luukkainen, Leo Leppänen, Nea Pirttinen, Vilma Kangas, Jarmo Isotalo and Joni Salmi.

Lastly, my deepest thanks go to my friends and family: my brother Antti, who is my best friend and a great colleague; my mom Eeva, whose scientific worldview has greatly influenced my own; and my loving partner Irma, who has always been there for me when I needed her the most.

Helsinki, October 2019
Juho Leinonen

Contents

1 Introduction **1**

1.1 Motivation and Research Questions 2

1.2 Publications and Contribution 5

1.3 Structure of the Dissertation 6

2 Background **9**

2.1 Using Data in Computing Education 9

2.2 Systems that Collect Keystroke Data 11

2.3 Keystroke Dynamics 13

2.4 Automatic Plagiarism Detection in Programming 14

2.5 Keystroke Data as Open Data 15

3 Research Approach **19**

3.1 Context and Data 19

3.2 Methodology 20

4 Results **25**

4.1 Inferring Programming Performance and Experience from Keystroke Data 25

4.2 Plagiarism Detection and Authorship Attribution Based on Keystroke Data 26

4.3 Anonymity and Information 29

5 Discussion **33**

5.1 Relationship Between Performance and Typing 33

5.2 Identifying Students Based on Typing 34

5.3 Detecting Plagiarism from Keystroke Data 36

5.4 Applications Outside Education 37

5.5 Open Data and Anonymity 37

5.6 Granularity of Data 38
5.7 Limitations 39

6 Conclusions and Future Work 41
6.1 Revisiting the Research Questions 41
6.2 Key Contributions 42
6.3 Future Work 44
6.3.1 Identification Based on Typing 44
6.3.2 Keystroke Data De-identification 44
6.3.3 Inferring Information from Keystrokes 44
6.3.4 Plagiarism 45
6.3.5 Other Research Directions for Keystroke Data . . . 45

References 47

Original Publications

This thesis consists of six original peer-reviewed publications and an introduction to the articles. The articles are referred to as Article I, II, III, IV, V, and VI in this thesis and are included at the end of the thesis. The following articles are included:

- **Article I** Juho Leinonen, Krista Longi, Arto Klami, and Arto Vihavainen. "Automatic Inference of Programming Performance and Experience from Typing Patterns." In Proceedings of the 47th ACM Technical Symposium on Computing Science Education, pp. 132-137. ACM, 2016.

- **Article II** Arto Hellas, Juho Leinonen, and Petri Ihantola. "Plagiarism in Take-home Exams: Help-seeking, Collaboration, and Systematic Cheating." In Proceedings of the 2017 ACM Conference on Innovation and Technology in Computer Science Education, pp. 238-243. ACM, 2017.

- **Article III** Krista Longi, Juho Leinonen, Henrik Nygren, Joni Salmi, Arto Klami, and Arto Vihavainen. "Identification of Programmers from Typing Patterns." In Proceedings of the 15th Koli Calling Conference on Computing Education Research, pp. 60-67. ACM, 2015.

- **Article IV** Juho Leinonen, Krista Longi, Arto Klami, Alireza Ahadi, and Arto Vihavainen. "Typing Patterns and Authentication in Practical Programming Exams." In Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education, pp. 160-165. ACM, 2016.

- **Article V** Petrus Peltola, Vilma Kangas, Nea Pirttinen, Henrik Nygren, and Juho Leinonen. "Identification Based on Typing Patterns Between Programming and Free Text." In Proceedings of the 17th Koli Calling Conference on Computing Education Research, pp. 163-167. ACM, 2017.

- **Article VI** Juho Leinonen, Petri Ihantola, and Arto Hellas. "Preventing Keystroke Based Identification in Open Data Sets." In Proceedings of the Fourth (2017) ACM Conference on Learning @ Scale, pp. 101-109. ACM, 2017.

Chapter 1

Introduction

In the last few decades computers and software have become pervasive throughout society. Technological innovations such as the Internet and the World Wide Web have diversified education. While blackboards and chalk are still widely used, modern classes can utilize technological elements like clicker questions [10] and live demonstrations [24] in teaching. However, more recently, an even bigger change has started happening as many courses are being only offered online [52]. Being fully online has allowed increased student intakes to courses. The largest online courses, often called Massive Open Online Courses (MOOCs) can have tens or even hundreds of thousands of students [15]. This change has happened as online courses do not have the same physical limitations like the size of the lecture hall, and with advances in automatic assessment [4, 18, 30], students can get immediate automatic feedback on their progress [46].

The materials used to study have changed as well. Having the learning materials online has allowed many advantages that traditional course books cannot have. For example, online materials can have interactive elements such as visualizations [76, 77] and embedded quizzes [13, 28]. Additionally, the way students use these materials can be tracked. This has enabled many new areas of research.

Data collected from online materials has been used, for example, to improve learning materials [50], predict students' success [2], and visualize students' progression in the material [32]. As utilizing data to improve education has become more common, the data collected from learning materials has become more and more fine-grained.

The gradual increase of more fine-grained data collection can be observed in the context of programming courses as well. Traditionally, students have returned solutions to assignments, which the instructor then grades. More recently, intermediate solutions have been collected for analysis. This is often done by collecting a snapshot of the source code at different stages of the programming process, for example when students compile, run, or test their programs [44, 63]. Some systems even collect each keystroke students type while they are programming [40, 63].

In this thesis, we study what benefits there are to collecting data at the keystroke level in programming courses. Here and later in this work, "we" refers to either the author or the author and his collaborators depending on the context.

1.1 Motivation and Research Questions

Altogether, the overarching theme of this thesis is *"What benefits are there to collecting keystroke level data in programming courses?"* This theme is analyzed through multiple research questions, which are detailed in this section.

The field of study where this research falls into is Computing Education Research [74]. The field is interdisciplinary and utilizes methods and practices from at least computer science, educational sciences, psychology, and statistics. The research outlined in this thesis focuses on using machine learning methods and statistics to infer information from keystroke data collected in programming courses.

A previous study on keystroke data collected from programming by Thomas et al. [82] found that how fast students type certain character pairs correlates with their performance in an exam. Since programming performance can be inferred from keystroke data, it could also be possible to infer programming experience of students. Such information would be useful for many purposes, for example to estimate students' proficiency post hoc or validating questionnaire answers about previous programming experience of students. A post hoc analysis might be necessary if a background survey was not answered by students. Additionally, survey answers are based on self-evaluation, and thus might not be comparable between students – one student might consider themselves a novice with a hundred hours of programming experience, while another might consider themselves an expert. This leads to the first research question of this work:

- RQ1. How well can students' programming performance and previous programming experience be inferred from keystroke data?

Based on our results in Article I, keystroke data can be used to infer students' programming performance and experience. This yields sanguine expectations that other information could be inferred from keystroke data as well.

While online courses have many benefits such as being able to teach a huge number of students at the same time, there are also problems. Students often work alone, and can be thousands of kilometers away from the institute and the teacher who organize the course. This leads into a situation where it is often hard for students to get help as they have to rely only on online resources such as chat rooms. Additionally, these courses usually still have deadlines that have to be met. Students might also see online courses as less serious compared to traditional courses, and students usually only need an email address to sign up. This is evident by the fact that MOOCs have a lot of dropouts with average completion rates of around 10% [41]. All this contributes to a common issue that plagues especially online courses: plagiarism. Combatting plagiarism is especially important in courses with high rewards as in some cases, online courses have partially replaced traditional entrance examinations [51, 84].

Plagiarism is particularly problematic in programming [71]. Students are encouraged to work together, and at the same time prohibited from submitting the exact same answer to exercises. Using external libraries is advocated and it is one of the best practices in industry: you should not reinvent the wheel. This leads into a situation where it can be hard, especially for novice programmers, to draw the line between fair use and plagiarism of other people's source code. For the teacher, detecting plagiarism can be hard as the source codes for exercises are naturally more similar to each other when the programs are solving the same problem compared to, for example, essays in natural language. Thus, it might be hard to say conclusively whether similarities have happened naturally or due to plagiarism.

With keystroke data, it is possible to look into a student's programming process as the whole path the student took from the beginning to the end of an exercise can be followed keystroke by keystroke. Using keystroke data to look into students' programming processes to detect plagiarism early would be beneficial as then the educator could intervene and guide the student towards better study techniques. The second research question of this work is:

- RQ2. How can keystroke data help with detecting plagiarism and with source code authorship attribution in programming courses?

In Article II, we study how the processes students take while they are programming could be used to detect plagiarism. While our results in Article II are promising, and show that keystroke data is very helpful for automatically detecting plagiarism, there are cases where the methods of that study will not work. The methods there are effective in combating simple cases of plagiarism, where the student either directly or indirectly copy-pastes another student's work. But what if the student has someone else complete the whole exercise for them? Methods that rely on copy-pasting or comparing students' processes do not work in that case, since all of the source code is new. An interesting question is whether we could detect when the person behind the keyboard has changed.

Fortunately, there have been studies which show that a person's typing pattern can be used to identify them [57]. Identification done this way is based on the rhythm of typing, that is, *keystroke dynamics*. The previous studies on the topic have mainly studied identification within the context of writing English or another natural language. If something similar would be possible in the programming context, we could use keystroke dynamics to detect cases of plagiarism where a student has a friend complete whole exercises for them.

In Article III, we study whether identification based on typing works in the programming context and how the amount of data affects identification accuracy. In Article IV, we refine our identification methodology by studying the amount of features required for identification, and how the context of the data affects identification accuracy. More specifically, we study whether we can identify students completing a programming exam based on data from exercises, and whether the exam type (lab versus take-home) affects the accuracy. Lastly, in Article V, we extend the context aspect by studying whether it is feasible to identify students when the context of the text changes from programming to natural language.

Our results from Articles III-V show that students in programming courses can be identified quite well based on their typing. This means that keystroke dynamics can be used to enhance plagiarism detection. However, this also means that keystroke data gathered from programming is sensitive as people can be identified from it. For example, the European Commission states that "biometric data for the purpose of uniquely identifying a natural person" is sensitive and thus has specific requirements within the context of the GDPR legislation[1]. In addition to possible legal troubles, this is also problematic for researchers. Is it ethical to share such data to other researchers? This is especially problematic as open data is becoming more and more common, with some publishers even requiring any published studies to publish data as well. This dilemma leads to the final research question of this work:

[1]https://eur-lex.europa.eu/legal-content/EN/TXT/HTML/?uri=CELEX:32016R0679

- RQ3. How could privacy and information be balanced in keystroke data?

The results of Article VI show that there are some methods that can be used to prevent keystroke-based authentication while retaining other useful information in the data. However, other possible methods for identifying people in the data possibly exist and thus further research is needed on balancing privacy and information in keystroke data.

1.2 Publications and Contribution

This dissertation includes and is based on six original peer-reviewed publications. All of the publications and the studies have been a joint effort where the author has been at least an equal contributor. The articles and the author's contributions are outlined below:

- **Article I** Juho Leinonen, Krista Longi, Arto Klami, and Arto Vihavainen. "Automatic Inference of Programming Performance and Experience from Typing Patterns." In Proceedings of the 47th ACM Technical Symposium on Computing Science Education, pp. 132-137. ACM, 2016.

  Article I describes an experiment where students' programming proficiency and previous programming experience were inferred based on their typing. The candidate, together with the second author, led the data analysis and contributed equally to the writing of the article.

- **Article II** Arto Hellas, Juho Leinonen, and Petri Ihantola. "Plagiarism in Take-home Exams: Help-seeking, Collaboration, and Systematic Cheating." In Proceedings of the 2017 ACM Conference on Innovation and Technology in Computer Science Education, pp. 238-243. ACM, 2017.

  Article II presents a study where different ways of automatically detecting plagiarism in take-home exams are examined. Possible plagiarists were identified based on their programming process during a take-home exam, and were interviewed subsequently. The candidate contributed to the design of the study and co-wrote the article.

- **Article III** Krista Longi, Juho Leinonen, Henrik Nygren, Joni Salmi, Arto Klami, and Arto Vihavainen. "Identification of Programmers from Typing Patterns." In Proceedings of the 15th Koli Calling Conference on Computing Education Research, pp. 60-67. ACM, 2015.

Article III outlines an investigation in identifying programmers based on their typing. The candidate, together with the first author, led the data analysis and contributed equally to the writing of the article.

- **Article IV** Juho Leinonen, Krista Longi, Arto Klami, Alireza Ahadi, and Arto Vihavainen. "Typing Patterns and Authentication in Practical Programming Exams." In Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education, pp. 160-165. ACM, 2016.

  Article IV is a continuation of Article III, and examines identification of programmers in exam conditions, which could be used to guarantee that the same person has completed the course assignments and the exam. The candidate was the lead author responsible for the design of the study, data analysis and writing.

- **Article V** Petrus Peltola, Vilma Kangas, Nea Pirttinen, Henrik Nygren, and Juho Leinonen. "Identification Based on Typing Patterns Between Programming and Free Text." In Proceedings of the 17th Koli Calling Conference on Computing Education Research, pp. 163-167. ACM, 2017.

  Article V explores how the text being written affects the accuracy of identifying the person typing. The candidate was responsible for outlining the design and methodology of the study, and supervised the data analysis and writing.

- **Article VI** Juho Leinonen, Petri Ihantola, and Arto Hellas. "Preventing Keystroke Based Identification in Open Data Sets." In Proceedings of the Fourth (2017) ACM Conference on Learning @ Scale, pp. 101-109. ACM, 2017.

  Article VI discusses the balance between anonymity and information through a study on how typing profiles could be anonymized to prevent identification of the people in the data while retaining some useful information in the data. The candidate was the lead author responsible for the design of the study, data analysis and writing.

1.3 Structure of the Dissertation

The structure of the dissertation is as follows. In Chapter 2, we discuss prior work on using data in computing education and examine using keystroke data in more detail. In Chapter 3, we detail our research approach, outlining and discussing the methodological choices made to answer the research questions of the thesis. Chapter 4 presents the main findings of the included articles. The results are presented in sections that are based on the research questions of the thesis. The results are then discussed in Chapter 5. Additionally, limitations of the work are presented. Chapter 6 concludes this work by revisiting the research questions and outlining key contributions. Lastly, potential future research directions for keystroke data are presented.

Chapter 2

Background

The theme of this thesis is studying how keystroke data can be beneficial in programming courses. In this section, we present a brief overview into topics related to this theme. More detailed descriptions of previous related studies are within the articles included in this thesis.

First, in Section 2.1, we discuss using data in computing education in general. Many types of data can be collected from the learning process, with keystroke data being one of the possible types. Then, in Section 2.2, we examine previous work on keystroke data in the context of programming by presenting a few systems that have been used in computing education to collect keystroke data. In Section 2.3 one particular use case of keystroke data is analyzed in more detail: keystroke dynamics, which is studying the rhythm of typing. Keystroke dynamics has been used, for example, to identify people. In the educational context, identifying the person typing can be used to authenticate users completing online courses. Then, in Section 2.4, automatic plagiarism detection in programming is discussed. Keystroke data presents unique opportunities for automatic plagiarism detection: for example, by reconstructing the programming process keystroke by keystroke, trivial plagiarism by copy-pasting is easily detected. Lastly, in Section 2.5, we discuss having keystroke data as open data. Since keystroke data can be used to identify the person typing, having keystroke data as open data can be problematic if the data should be anonymous.

2.1 Using Data in Computing Education

Data collected from the learning process of students can be used to improve education. The learning process can include many types of activities: for example assignments, lectures, and studying course material. The field of study in which using data to improve education is researched has been called, for example, learning analytics, educational data mining, data-driven instruction or data-driven education. There has been a lot of research and multiple literature reviews on the topic [5, 61, 69].

The activities and materials from which data has been collected in education has sometimes been called smart learning content. Brusilovsky et al. studied smart learning content, which they define as interactive web-based learning content [9]. These include program visualization tools [25, 68, 76, 77], automatic assessment tools [4, 18, 30], and other interactive tools. One of the advantages of smart learning content is the ability to collect data about how students use these tools [50, 70]. Brusilovsky et al. note that data collected from smart learning content can have benefits to many stakeholders: instructors, students, researchers, smart learning content authors and the smart learning content platforms themselves.

Ihantola et al. [31] conducted a literature review on educational data mining and learning analytics in programming. While Brusilovsky et al. [9] focused on learning content, Ihantola et al. focused more on the data collected from programming. They studied the types of data that have been collected and analyzed in programming courses, focusing also on whether studies have been replicated by other researchers. They found that more studies used high-level data – such as submission data – than low-level data – such as keystroke data. Additionally, most of the data sets used have not been open. Ihantola et al. found that the most common way to collect data from the programming process is to instrument the programming environment with automatic data collection.

Data collected from the learning process can be used to improve education in many ways. For example, data about how a student is performing in a course can be used to find students who require additional help [26], which allows instructors to stage an intervention [81]. An intervention can take many forms: it could be additional assignments, additional teaching opportunities, or simply showing the student a visualization about their progress. Many students struggling with a certain concept might indicate that there is something wrong with the learning material itself, and thus data can be used to improve the learning material as well [50].

In the context of programming, one of the programming-specific ways to collect data about a student's learning process is to use an Integrated Development Environment (IDE) in which students work on course assignments. Using an IDE allows instructors and researchers to easily collect data about the programming process of a student since many IDEs can be modified to support data collection, for example, by customized plugins, which has been the most common way to collect data from programming [31]. Depending on the IDE, different types of data can be collected. For example, BlueJ, a popular IDE used in programming courses, collects anonymized versions of students' source code as well as IDE actions such as compiling, running and testing code [8].

Hundhausen et al. [29] recently presented a process model for IDE-based learning analytics. The process model they developed includes four stages: 1) collecting data, 2) analyzing the data, 3) designing an intervention, and 4) delivering the intervention. They reviewed different systems that are used to collect data from IDEs and studies that have used IDE-based data to stage interventions to students. One of the call-to-action items they urge future research to tackle is privacy of data collected from IDEs, which we discuss later in this thesis.

One aspect of data is the granularity of the data. When considering programming, keystroke data has been noted as the most fine-grained type of data usually collected from programming [9, 31] while submissions to assignments [31] or student level data such as age [9] have been noted as the most coarse-grained. However, one could argue that for example physiological data [3] is even more fine-grained than keystroke data, since there could be multiple physiological data points between two keystrokes. One of the benefits of fine-grained data is that it is possible to get a more detailed view into the process a student took while programming [83].

2.2 Systems that Collect Keystroke Data

Keystroke data is data that is collected every time a key on the keyboard is pressed. Generally, keystroke data can be collected with either a hardware or a software keylogger. Academic studies have mostly used software keyloggers as they do not require physical components and can be installed remotely to a computer. Additionally, software-based keyloggers can easily be configured to only capture keystrokes within a specific program, which is good from a privacy perspective.

Many systems that collect keystroke data have been developed over the years. We will discuss a few that collect data from programming as that is the context of our studies. Many systems collect data from the programming process, but only a few explicitly state that keystroke level data is collected. For example, while Web-CAT [19], Marmoset [79] and BlueJ [44] all collect data, they do not collect data at the keystroke level. The most fine-grained data that Web-CAT and Marmoset collect are snapshots when students save their source code, and for BlueJ the most fine-grained snapshots are edit events where multiple edits to a single line of code are condensed into a single snapshot.

One example of a system in which keystroke data is collected from programming is CloudCoder [62], which is a web-based programming assignment system. CloudCoder is designed to support short programming exercises and the developers of CloudCoder provide an open repository of assignments that are free to use. Students can submit the exercise directly in CloudCoder, after which the submitted solution is tested against a test suite. Students are shown the passing and failing test cases in the browser to help them debug their program if all the tests do not pass. The data collected in CloudCoder has been used to predict students' success in introductory programming courses [78].

The system that is used for collecting keystroke data in this thesis is Test My Code (TMC) [63]. TMC is a service that facilitates students' learning in many ways. It allows easy management of exercises: students can download exercise templates and return completed exercises through TMC. In addition, TMC can be used for automatic testing of students' programs against a test suite that tests whether the students' programs work correctly according to the specifications of the exercise. An instructor can define as many test cases as they wish. If the program passes the tests, students can get points for the completed exercise. For failing tests, the instructor can define feedback to be given to the student. While originally developed for Java programming, TMC now supports a multitude of programming languages such as Java, Python, and C. Students can return their exercises for testing in multiple ways. TMC has a web UI that allows zip-files to be uploaded. Additionally, there is a command line interface and plugins for a couple of popular IDEs such as NetBeans and IntelliJ. The IDE plugins add the functionality of TMC directly into the IDE by implementng buttons for testing and submitting assignments.

In addition to providing an easy way to test and return exercises, TMC also collects data from the process the students took while programming if the students use one of the plugins that support TMC's data collection (for example the plugin for NetBeans). Data is collected based on actions that the students take within the IDE as well as every time the contents of a source code file change, for example when students write a character or remove a character. Thus, the data is at the keystroke level. Data is also collected every time students run or test their programs, as well as when other IDE actions such as debugging are conducted.

There most likely exist many other systems that collect keystroke data from programming, but where the fine-grainedness of the collected data is not specified. Additionally, there are some studies [22, 43] where keystroke data from programming has been collected, but the specifics of the system being used are not publicly available.

2.3 Keystroke Dynamics

A much studied research area that utilizes keystroke data is keystroke dynamics [23, 36, 38, 57, 86]. In keystroke dynamics, the rhythm of typing on a keyboard is used to build typing profiles of people. Keystroke dynamics has been mostly studied from the point of view of identifying someone based on the uniqueness of the rhythm of typing [38].

Different types of data can be collected from typing. The simplest keyloggers collect just the keys pressed, while more sophisticated keyloggers collect additional information such as the time of the keypresses or the pressure of keystrokes [59]. Most commonly, keystrokes and their timings are used, since collecting those does not require any special hardware as a traditional keyboard can be used, whereas collecting information such as the pressure of keystrokes is not possible with a traditional keyboard. This keystroke data is processed into typing profiles.

Typing profiles can be used to identify the person using the keyboard as it has been found that the way a person types is a biometric identifier [23, 36, 38]. Biometric identifiers are characteristics of a person that can be used to uniquely identify them [33]; other typical biometric identifiers include a person's fingerprint, the iris of the eye, and handwriting. Most commonly, typing profiles include *digraph latencies*, also known as character pair latencies [64]. Digraphs are character pairs. For example, the word *pair* has three digraphs: *pa*, *ai*, and *ir*. Average digraph latencies are the average times it takes the person using the keyboard to type different digraphs that occur during typing. Depending on the context, different ways of calculating the average time have been used. This is often due to technical details of the software / hardware used to collect the data: some methods calculate the latency based on when a key is pressed, others calculate it based on when a key is released, and some rely on visible changes to text [53, 64]. In Article III, we study identifying someone based on their typing in the context of programming.

Even though the way a person types is a biometric identifier, it can be affected based on the context of the typing. For example, some studies have found that the keyboard being used affects the reliability of identification [86]. Similarly, the type of text being written has been found to affect identification [57]. In Articles IV and V we analyze how the context of typing affects typing-based identification. In Article IV we analyze how identification changes between assignments completed at home and programming tasks in exams both at home and at the university. In Article V we examine whether it is possible to identify someone typing Finnish based on typing profiles built from programming data and vice versa.

In addition to identifying the person typing, keystroke dynamics has been used to identify attributes related to the writer. There have been some studies on recognizing emotional states based on typing patterns [6, 21, 42, 43]. The results of these studies indicate that the emotional state of the typist affects their typing patterns, which allows estimating the current emotional state of the typist based on how they type. Additionally, demographic factors such as gender have been inferred by keystroke analysis [7]. Keystroke dynamics have also been used to predict programming performance [82]. The results indicate that more skilled programmers type differently than less skilled programmers. In Article I we partially replicate the study in [82] and also investigate whether prior programming experience can be inferred based on typing patterns. Considering the R.A.P. taxonomy presented in [31], our study is a reproduction as it involves new researchers investigating new data with new methods to reproduce the results of the previous study.

2.4 Automatic Plagiarism Detection in Programming

Many tools have been developed to automatically detect plagiarism in programming. These tools include, for example, JPlag [65] and MOSS[1]. Most of the tools rely on comparing source code files to one another to find whether two files are similar enough to warrant plagiarism concerns. The tools often employ methods to counteract typical measures plagiarists take to hide and disguise plagiarism. For example, the tools might only compare the structure of the program, and do not take the naming of variables into account, since plagiarists typically try to disguise plagiarism by changing variable names.

Plagiarism is common when learning programming [35]. One of the reasons might be that the definition of plagiarism seems to not be very clear to students [11, 34]. On one hand, students are often encouraged to work together and to use external libraries – to not “reinvent the wheel”, but at the same time students are expected to not copy-paste source code

[1] https://theory.stanford.edu/~aiken/moss/

without attributing it to the original author. Traditionally, there has not been a standard format for attributing borrowed code easily [75], which exacerbates the problem. Additionally, as programming is done on the computer, copying someone else's code is trivial compared to having to handwrite plagiarized answers. This is exacerbated by the wide availability of example code: third party sources such as Stack Overflow[2] are often used to find solutions with code examples to common problems students can encounter, and services such as GitHub[3] are used to host open source projects from which students can potentially copy code.

In Article II we study automatically detecting excessive collaboration in a take-home exam. We use keystroke data to reconstruct the process students' took while constructing their solutions and compare the processes to identify cases where students may have plagiarized or collaborated with others, which was not allowed during the take-home exam.

2.5 Keystroke Data as Open Data

Traditionally, research data has been closed and only specific people have had access to it. Nowadays, there is a push to have open data [45]. The most extreme example is to have all of the data available to anyone interested, although there have been other examples where data is only partly open: for example, some features have been removed (for instance removing identifying information due to privacy concerns [16]), or data is only shared for pre-specified purposes (for instance data might be released only for non-commercial purposes [60]).

There are many benefits to having open data in academia. Open data makes it easy for third parties to replicate studies using the same data to verify results, which increases the transparency of science [1, 31]. In addition to verifying previous results, open data enables other researchers to conduct novel studies, which advances science.

However, open data also has its downsides. In the case of data related to people, one of the major problems is related to privacy. Data can easily include personally identifiable information, and it is not always obvious which features of the data could be used to identify people. Identifiers in data can be split into two categories: explicit identifiers and quasi-identifiers [14,80]. Explicit identifiers are features such as a person's full name or their social security number, which can be used by themselves to identify a person. Quasi-identifiers on the other hand are features that by themselves can not

[2]https://stackoverflow.com/
[3]https://github.com/

be used for identification, but may form an identifier when combined with other features. For example, age is not an explicit identifier, since many people share the same age. However, if you combine age with a postal code, it is possible that in a certain postal area, there is only a single person with a certain age. Thus, combined together, age and postal code may form an identifier, and thus both age and postal code are quasi-identifiers. The more quasi-identifiers you combine, the more likely it is that they become an identifier.

As an example of what can go wrong when data is released openly, when Netflix[4] released data about users' movie ratings, researchers were able to identify people as well as infer private information about them based on the data [58]. They used another publicly available data set – movie ratings in the Internet Movie Database (IMDb)[5] – and combined data from both. If a user had rated a movie at around the same time in both services, and this occurred multiple times in the data, it was determined to be fairly likely that the person rating the movies is the same in both data sets. The problem here is that when a user is giving a rating in Netflix, they most likely think that the given rating will remain private. However, the same person could have a public IMDb profile and avoid rating controversial movies publicly. Based on how users' rated certain movies, the researchers were able to infer political preferences and even sexual preferences of the users.

Keystroke data can be used to identify the person who did the typing [38], more specifically the combination of keystrokes and their timings can be used to build a typing profile that is unique to the person typing. Thus, keystroke timings are a quasi-identifier – you need to combine many of them to get an accurate typing profile. This essentially means that keystroke data contains personally identifiable information – the keystrokes and their timings.

Since keystroke data can be used to identify people, having keystroke data as open data is questionable. For example, if keystroke data collected in a programming course was released openly, someone else with similar data could connect the data sets and gain information about the person. Keystroke data has been shown to be usable for identifying emotional states [21, 42]. Thus, a possible, if unlikely, example could be that someone with depression has participated in a programming course, and later applies for a job at a company. If the company also has keystroke data about the applicant (for example collected during a technical interview), they could

[4]https://www.netflix.com

[5]https://www.imdb.com/

connect the data sets and potentially gain sensitive medical information about the past emotional states of the applicant. Based on this, it might not be ethical or fair to have keystroke data as open data. In Article VI we examine anonymization of keystroke data. We investigate whether we can find a balance between anonymity and information in a way where the people in the data can no longer be identified based on keystroke dynamics, while retaining other information related to keystroke dynamics in the data so that the data still has value to researchers.

Chapter 3

Research Approach

In this chapter, we describe the research approach of this thesis. We first outline the context of the research conducted and the data used in the studies in Section 3.1. We then explain the methodological choices of the thesis and the articles in Section 3.2.

3.1 Context and Data

The studies in this thesis have been conducted with data collected from multiple iterations of two introductory programming courses held at the Department of Computer Science of the University of Helsinki in Finland. The programming language taught is Java. Together, the courses last for 14 weeks (7 weeks each) and cover traditional introductory programming topics such as variables, printing output, reading input, objects, classes, interfaces, etc.

The course pedagogy relies on having many small automatically assessed exercises instead of larger projects. The number of exercises has varied a little based on the course iteration, but there have typically been around 10-30 exercises a week in the courses. With small exercises, students get the feeling of accomplishment early and often [85]. Additionally, with small exercises, students get feedback earlier and more often than with larger exercises, and are more likely to start working on them early compared to larger more complex exercises [17]. Small exercises also guarantee that students get repeated practice on important concepts, which has been shown to increase long-term retention of information [39].

How students are assessed at the end of an introductory programming course varies a lot between institutions [72, 73], but electronic examinations can have benefits such as being able to use automatic assessment and allowing students to debug code [67]. Thus, we have decided to use electronic examinations as the end of course assessment. The electronic exams contain assignments similar to those completed as weekly exercises in the course.

Students in the courses use an Integrated Development Environment (IDE) with a custom plugin that allows exercise management. The courses use an IDE starting with the very first exercise. This has been done so that students learn to use professional tools for programming at the same time they are learning the concepts. The IDE of choice here is NetBeans. NetBeans was chosen as it is open-source and commonly used for Java development. The custom plugin we use for NetBeans to manage exercises is called Test My Code [63] and is discussed in Chapter 2 under Section 2.2.

The data collected from Test My Code consists of different types of events students take whilst programming in the IDE. The most relevant data for this thesis are edit events, which are collected when students edit source code in the IDE. The edit events are based on comparing the source code before and after a student edits it. Thus, they usually only contain a single character addition or deletion, which is the case when students are writing source code. If students remove larger blocks of text, or copy-paste code from somewhere else, the edit events can include multiple characters in these cases.

3.2 Methodology

The overarching theme of this work is *"What benefits are there to collecting keystroke level data in programming courses"*. This question is examined through a series of experiments where different uses of keystroke data are studied. The used data has been gathered over the years and due to this, most of the studies use post hoc quantitative analyses such as different machine learning methods. Additionally, in all of the studies, the focus is on the usefulness of keystroke data and its applications for education.

The granularity of collected data affects the usefulness of the data, but also the resources needed to collect the data. From the usefulness perspective, more fine-grained data should always be strictly more beneficial as less fine-grained data can be obtained by filtering more fine-grained data. However, more fine-grained data requires more disk space as well as uses more bandwidth when being transmitted. A less obvious disadvantage is that more fine-grained data is also likely to contain more personally identifiable information, and thus requires more careful handling.

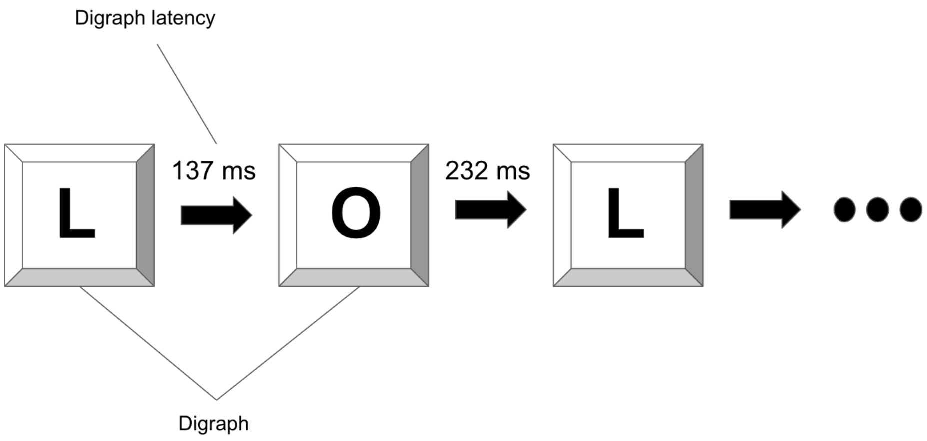

Figure 3.1: An illustration of digraphs and digraph latencies. A digraph is a pair of two characters, for example “L” and “O” as in the picture. A digraph latency is the time between two sequential keypresses, which form the digraph.

Table 3.1: An example typing profile built based on Figure 3.1. The typing profile consists of the average times it took the person typing to type two different digraphs, “LO” and “OL”.

	Digraph	Average digraph latency
Digraph1	L->O	137 ms
Digraph2	O->L	232 ms

We have decided to collect data at the keystroke level, that is, collecting every keystroke of students while they are programming. More fine-grained data should allow a more accurate view into the learning process of students, which could have benefits to both researchers and educators. Seeing the whole process instead of only the final product could provide insight on which parts of the process are hard for students, and which parts are easy. This information could be used to – for example – stage appropriate interventions where students are given more help on certain topics.

Many of the experiments we conduct are based on analyzing students’ typing profiles. The typing profiles consist of the average times it takes students to write certain character pairs, that is, digraphs. Figures 3.1 and 3.2 contain two examples of keystroke chains and the latencies between those keystrokes, and Tables 3.1 and 3.2 contain the resulting typing profiles respectively.

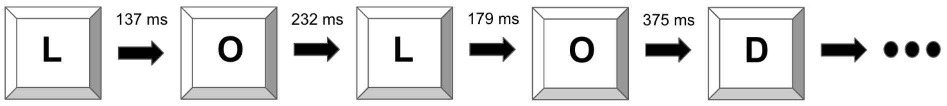

Figure 3.2: A sequence of five keystrokes: "L", "O", "L", "O" and "D".

Table 3.2: An example typing profile built based on Figure 3.2. The typing profile consists of the average times it took the person typing to type three different digraphs, "LO", "OL" and "OD".

	Digraph	Average digraph latency
Digraph1	L->O	158 ms
Digraph2	O->L	232 ms
Digraph3	O->D	375 ms

We study the benefits of keystroke data through three research questions. The methodologies used to study each question are presented here. A mapping between the research questions and the included articles is in Table 3.3.

- RQ1. How well can students' programming performance and previous programming experience be inferred from keystroke data?

In Article I, we study RQ1 by conducting a study where we replicate a study by Thomas et al. [82] who found that keystroke data can be used to infer students' programming performance. They divided digraphs into categories – for example numeric, alphabetic, browsing, etc. – based on the types of keys the digraphs consisted of. The results of their study indicate that the speed of writing numeric and "edge" (characters in the digraph are in different categories, but are not browsing characters) digraphs correlates with students' scores in a written test: those who wrote these digraphs faster performed better in the test. In our replication, we only have data for some of the categories as we do not have data on browsing or control keys due to limitations of the software used to collect the data in our studies.

Additionally, we extend Thomas et al.'s study by exploring using different machine learning methods to predict exam performance and students' previous programming experience based on students' typing patterns. We use the Random Forest and Bayesian Network machine learning classifiers to predict students' performance in a programming exam and whether students' have previous programming experience. We compare the results to random guessing to see whether keystroke data can achieve better prediction performance.

	RQ1.	RQ2.	RQ3.
Article I	x		x
Article II		x	
Article III		x	x
Article IV		x	x
Article V		x	
Article VI			x

Table 3.3: Mapping between articles and research questions.

- RQ2. How can keystroke data help with detecting plagiarism and with source code authorship attribution in programming courses?

In Articles II–V, we study RQ2. We use triangulation, that is, conduct multiple experiments where we study different ways of utilizing keystroke data to automatically detect plagiarism and to identify the programmer. There exist many methods to detect plagiarism based on final submissions to programming tasks, for example JPlag [65]. However, with keystroke data, it is possible to reconstruct the programming process keystroke by keystroke, and thus we focus on methods that explicitly rely on the fine-grained nature of the data.

In Article II, we examine automatic ways to detect plagiarism and excessive collaboration in take-home exams. We first identify possible cases of plagiarism and excessive collaboration based on multiple factors. We analyze keystroke data collected during a take-home exam to analyze whether plagiarism could be automatically identified from the data. We base the analysis on examining the programming process and comparing students' processes to one another. A single student's process can indicate plagiarism if it contains abnormalities such as copy-pasting while two or more students having similar processes can indicate excessive collaboration.

In Articles III, IV and V we examine plagiarism from a different angle compared to Article II. In Article II, we focus on identifying programming processes that indicate plagiarism. In Articles III–V we instead focus on identifying the person completing the assignment. The methodology we use in Articles III–V is based on keystroke dynamics, that is, the rhythm of typing. Identification based on typing in our studies is conducted by building typing profiles of students in different settings and then comparing the typing profiles to find whether a student's typing profile in one setting matches the typing profile in the other setting. For example, in Article IV, we examine identifying a student in a take-home exam based on a typing profile built from assignment data. If the same student who completed the assignments attends the exam, the typing profile in the exam should be a close match to the typing profile in the assignments.

- RQ3. How could privacy and information be balanced in keystroke data?

In Article VI, we study RQ3 by conducting an experiment where we de-identify[1] typing profiles. Our goal is to find a balance between anonymity and information. This is based on the realization that when anonymity of data is increased, the informational value of the data is decreased. For example, for perfect anonymity, one could remove every possible feature of data that could be used to identify someone, but this would result into a lot of lost data. On the other hand, the more features there are in data, the more likely it is that someone could be identified based on the data if those features are quasi-identifiers. We focus on preventing identification based on typing and do not try to prevent identification altogether. Even if identification based on typing can be prevented, other ways of identification based on, for example, textual content or stylometry (the style of the text) could still be possible.

To explore the balance between anonymity and information, we conduct a case study where we examine how keystroke-based identification accuracy changes when we de-identify the typing profiles as a measure of how anonymous the data is. To measure informational value, we use the methodology used in Article I where typing profiles are used to classify students into novice and more experienced programmers. For identifying people, we use the methodology outlined in Articles III and IV. The goal of the case study is to find whether we can find a balance where students cannot be identified based on typing, but their programming experience can still be inferred. If such balance is found, we can say that the data is at least partially anonymized – against our keystroke-based identification approach – but the data still has value to researchers since students' programming experience can be inferred.

[1]De-identification is similar to anonymization, but the former term is more commonly used in academic studies. Anonymization as a term seems to imply that identifying people in the data is impossible, whereas de-identification implies that certain ways of identifying someone have been made harder.

Chapter 4

Results

In this chapter, we describe the main findings of the publications included in this thesis. Only the key findings are presented. For more thorough analysis, see the original publications at the end of the thesis. Sections 4.1, 4.2, and 4.3 present results related to research questions 1, 2, and 3 respectively. Each research question is reiterated in the beginning of their corresponding Section.

4.1 Inferring Programming Performance and Experience from Keystroke Data

The first research question of this thesis is *"How well can students' programming performance and previous programming experience be inferred from keystroke data?"* Based on our results in Article I, the programming experience and performance of students can be automatically identified based on their typing to some extent.

In Article I, we partially replicated a study by Thomas et al. [82] and found results that supported their findings. The main result of both studies is that students who perform better in programming based on exam performance are faster at writing certain digraphs. Both studies found that better performing students are faster at writing digraphs where the type of character changes (such as when going from a numerical key to an alphabetic key) – in our results, the correlation with exam scores was -0.227 and in Thomas et al.'s study the correlation was -0.276. Additionally, both studies found that the speed of writing numeric digraphs (where both characters are numbers) correlates with exam performance with correlations of -0.170 and -0.333 for our and Thomas et al.'s study respectively.

In addition to replicating Thomas et al.'s study, we examined classifying students into high and low performing students – over or under median exam score – based on their digraph latencies. We found a classification accuracy of 65% on the first week of the course with the accuracy reaching a little over 70% in the last week of the course compared to an accuracy of around 52% for a baseline majority classifier.

Lastly, we studied whether students' prior programming experience can be inferred from keystroke data. There are some digraphs which are common in programming and rare in natural language such as digraphs containing special characters. The results of the study show that these programming specific digraphs are the most telling of prior experience or skill in programming as experienced programmers typed these digraphs faster on average. An example of such digraph is *i+* which is often written in source code when incrementing a variable called *i* (*i++*). The difference in typing speeds of *i+* is visualized in Figure 4.1. The classification accuracy was around 75%, which is slightly better compared to a baseline majority classifier which always classifies a student as not having any programming experience, which had an accuracy of around 60%.

4.2 Plagiarism Detection and Authorship Attribution Based on Keystroke Data

The second research question of this thesis is *"How can keystroke data help with detecting plagiarism and with source code authorship attribution in programming courses?"* Based on our results in Articles II-V, keystroke data can help with detecting plagiarism and with source code authorship attribution mainly in two ways: 1) by reconstructing the programming process and looking for anomalies (Article II), and 2) by building typing profiles of students and using those for authorship attribution (Articles III-V).

In Article II we examined keystroke data collected from a take-home programming exam. Students were able to choose a suitable time to start the four hour long exam. Keystroke data allows a key-by-key reconstruction of the whole programming process and we found that reconstructing the processes students took to reach their solutions can be used to detect plagiarism. Firstly, from keystroke data, it is trivial to notice copy-pasting. Looking at the reconstructed programming process, copy-pasting shows up as a sudden influx of text. Secondly, comparing students' processes to one another can reveal processes that are similar. We found that similar processes can indicate excessive collaboration and plagiarism.

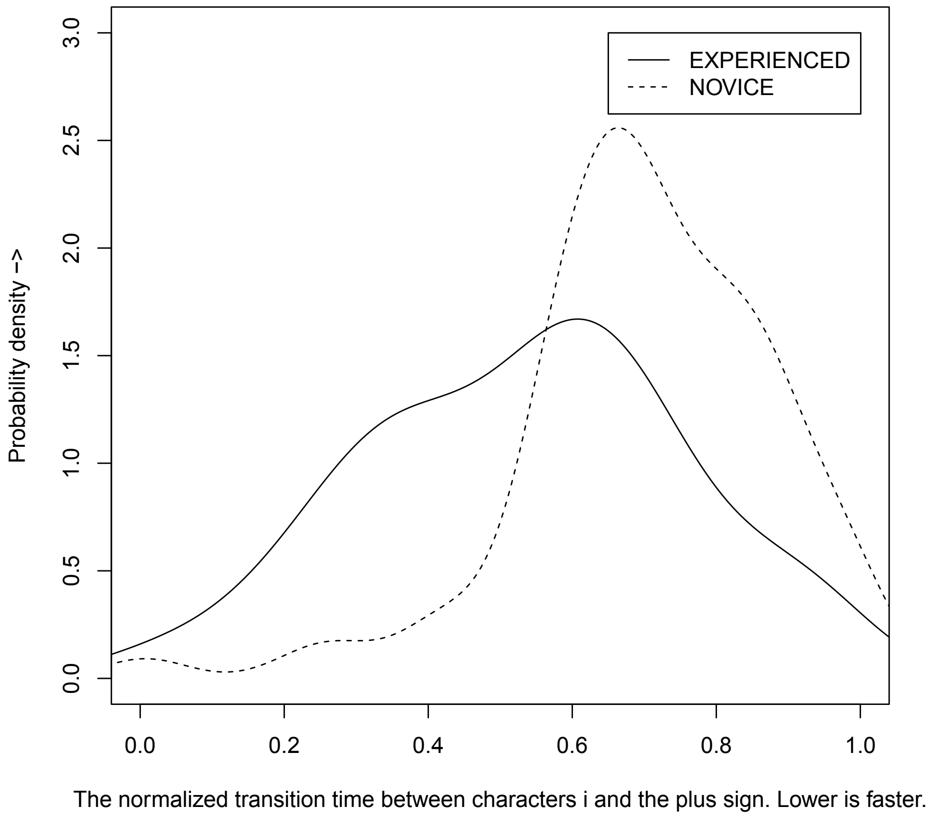

Figure 4.1: Smoothed probability density function of the times taken between pressing the characters i and + by novice and experienced programmers [49].

Table 4.1: The effect of the amount of data used to build a typing profile on identification accuracy. A typing profile was built with data from the “training weeks” and students were identified on the “identification week” [53].

Training weeks	Identification week	Students	Correctly identified	Accuracy
1	2	153	119	77.8%
1-2	3	153	126	82.4%
1-3	4	153	135	88.2%
1-4	5	153	135	88.2%
1-5	6	153	133	86.9%
1-6	7	153	146	95.4%

In Article III we studied using keystroke dynamics to identify programmers. Our findings indicate that programmers can be identified based on typing with a high accuracy – using data from the first six weeks of a course, students in the seventh week could be identified with a 95% accuracy. This essentially means that students in programming courses can be identified based on their typing when keystroke data is collected. Table 4.1 shows the identification accuracy based on how much data is used to build the typing profiles.

In Article III, we studied keystroke-based identification with data gathered from programming assignments. One limitation of that study was that all the data was related to assignments. Thus, it is not certain whether identifying a programmer based on typing is possible when the context is changed, for example if the programmer uses a different keyboard or if the context of the assignments is different. In Article IV, we studied whether data gathered from assignments could be used to identify a programmer in an exam. While the context is somewhat similar, for example in both the text that is written is source code, there are some differences as well. We studied both identifying students taking a take-home exam as well as students attending an exam at university premises. In both cases, it is likely that students are more stressed than during regular assignments as the exams are high-stakes and have a tight time limit. Additionally, when students are completing the exam at university premises, it is likely that they are using a different keyboard than during regular assignments. We found that identifying students in both types of exams is possible based on the data gathered from assignments with over 85% accuracy (with $n = 69, 61, 153, 128$ for two campus and two take-home exams respectively), but identification accuracies are somewhat lower than when the data comes from a single context where the identification accuracy was around 95% (with $n = 153$).

We further explored how changing the context affects identification accuracies in Article V. We gathered data from programming assignments, programming exercises in an exam, and essay questions in an exam. We studied how the accuracy of identification varies when the text that is written varies between programming and natural language. The results of the study indicate that identification is considerably more difficult but still possible to some extent when the text someone types is of different type than the text that was used to build the typing profile. We found that using data from 12 weeks of programming assignments to build typing profiles, 73% of students completing a programming exam assignment were correctly identified, while only 50% of students writing an essay answer were correctly identified. This indicates that the context of the writing (essay versus programming) does affect the identification accuracy. The 50% accuracy is still considerably higher than random guessing (which would have an accuracy of under 1%).

4.3 Anonymity and Information

The third research question of this thesis is *"How could privacy and information be balanced in keystroke data?"* Based on our results in Article VI, by de-identifying keystroke data, at least keystroke-based identification can be made less accurate while retaining some keystroke-related information in the data.

In our case study, we tried to find a balance where students' programming experience could still be inferred based on their typing (similar to Article I), but the person typing could not be identified anymore (similar to Articles III-V). Figure 4.2 shows how the identification accuracy and the programming experience classification accuracy changes depending on the amount of de-identification. The method used for de-identification here is dividing keystroke latencies into categories – very slow, slow, average, fast, and very fast – instead of having the exact average keystroke latency in the typing profile. In the article, we call these categories "buckets". The amount of categories depends on the amount of anonymization. For example, if data contains latencies between 10 and 750 milliseconds, and if the bucket size is 150 milliseconds, there are five buckets: 0-150, 150-300, 300-450, 450-600, 600-750 (see Article VI for details).

Based on our results, there is a de-identification point where students can no longer be identified reliably, but their programming experience can

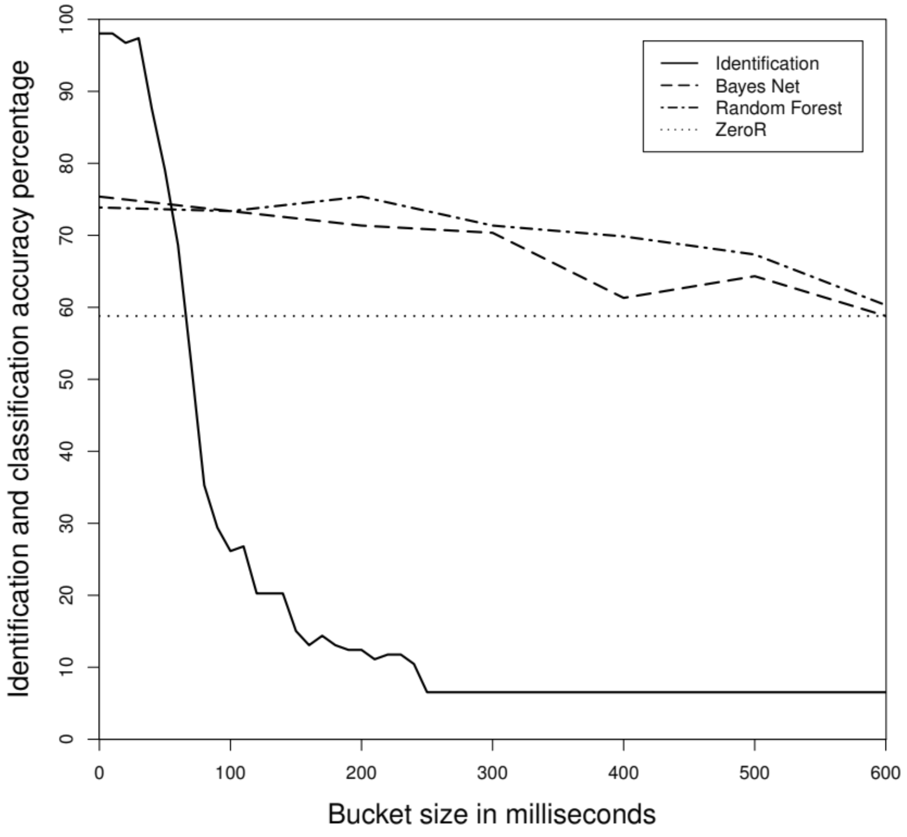

Figure 4.2: Identification (solid line) and programming experience (dashed lines) classification accuracy compared against increasing de-identification. Students' typing profiles were modified so that instead of having the exact average latencies for digraphs, they only had information on which category (bucket) each digraph belonged to (see Article VI for details). Programming experience classification accuracies are shown for three different classifiers: Bayesian Network, Random Forest, and the majority class classifier ZeroR. The x-axis represents bucket size and the y-axis expresses identification and classification accuracy [47].

still be inferred with an accuracy that is higher than simply guessing. For example, with the data used in Figure 4.2, a suitable “balance” could be at around x = bucket size = 300 milliseconds. At that point, the identification accuracy is quite low at around 7%, while the programming experience classification accuracy is at around 71% compared to around 59% with the majority classifier.

Chapter 5

Discussion

In this chapter, we discuss the results of this thesis. First, the relationship between performance and typing is discussed in Section 5.1. Then, results on identifying students based on typing are reviewed in Section 5.2. We then reflect on detecting plagiarism from keystroke data in Section 5.3. Section 5.4 discusses possible applications of the results outside of education and Section 5.5 discusses open data and anonymity. The fine-grained nature of the data is discussed in Section 5.6. Additionally, we present the limitations of this work in Section 5.7.

5.1 Relationship Between Performance and Typing

The results of our experiments indicate that the way students type tells about students' previous programming experience and their programming performance, which supports earlier findings by Thomas et al. [82]. It should be noted that experience and performance can be argued to be similar measures to one another: as you get more experience on doing something, you are likely to get better at it. On the other hand, there could be differences due to someone having natural talent at a skill, and thus learning it faster, in which case a more experienced person could still perform worse than a less experienced one. Both are detected the same way through typing: those with more experience, or more proficiency, in programming are faster at typing programming-related digraphs such as *i+* or *//*.

While previous experience can be collected with a questionnaire and performance can be inferred from assignment performance, the impact of our results is that if we do not have any other information than keystroke data, these attributes can be estimated from the data. For example, it is possible that we have keystroke data but a student did not complete a background questionnaire on programming experience, or it is possible there was no question about programming experience in any questionnaire. Furthermore, questionnaire answers are subjective. As an example, a background questionnaire on programming experience might ask students about the number of hours they have programmed before the course. If the topic of the course is Java, having a 100 hours of Java experience could be many times more beneficial than having 200 hours of PHP experience, but merely looking at questionnaire answers these differences might not be evident. Moreover, students might not answer questionnaires truthfully, for example in fear of appearing incompetent. Additionally, collecting keystroke data is passive and non-intrusive, so getting any information through it is good from the viewpoint of not having students spend time answering questionnaires.

Since programming performance can be partly inferred based on keystrokes, it is possible that keystroke data could be used to support decisions on whether an intervention, for instance additional assignments, should be given to a student. Using keystroke data to infer the performance of a student could also be beneficial in situations where the student has not yet submitted a solution, for example due to being stuck on the assignment. Furthermore, there could be open-ended assignments where the goals of the assignment are not clear, and thus for example automatic assessment is not possible. In these cases, keystroke-based performance inference could still be possible, while more traditional methods that rely on, for example, passing test cases would not be feasible.

When anonymizing typing profiles in Article VI, we found that the exact average typing speeds of digraphs are not necessary for inferring students' previous programming experience from typing. In fact, just specifying whether students are fast or slow at writing each digraph is sufficient. This is most likely due to the predictions being mostly based on how fast students are at typing programming specific digraphs. Thus, if enough programming specific digraphs are classified as fast instead of slow, the student is classified as likely having programmed before.

5.2 Identifying Students Based on Typing

Keystroke data can be used to build a typing profile of a student. In our studies, we decided to mostly use the information on how fast students typed certain character pairs – digraphs – while they were programming.

These average digraph latencies were found to be able to be used to identify students quite accurately: identification accuracies varied depending on different factors, but were consistently at least around 90%. To the best of our knowledge, these studies are the first in which keystroke-based identification has been studied in the context of programming.

Being able to identify a student brings many benefits. In our context, we have a massive open online programming course (a programming MOOC), where at the end of the course students can attend an exam and if they perform well enough, they are granted a study right to the University of Helsinki [84]. Our results show that with keystroke data we can check whether the typing profile the students had during programming assignments completed at home matches the typing profile of the student in the exam. With this information, it is possible to detect cases where the student who attended the exam is different from the one who completed the programming assignments. This type of authentication is not new and has been used for example by Monaco et al. [56] and Coursera [12,54]. However, our results confirm that this also works with programming.

We found that identification is also somewhat accurate when identifying students writing an essay based on typing profiles built from programming assignments and vice versa. This means that students can be identified even when the text students write is quite different (source code versus Finnish text). Additionally, we found that just using data from the most common 25 digraphs in the data is sufficient to reach identification accuracies that are high enough for quite reliable identification.

While we are able to achieve high identification accuracies, there exists the possiblity that a student is falsely identified as having cheated. Thus, keystroke-based identification should not be seen as an incontestable way of detecting cheating, but only used for example to raise a flag for manual checking of students' answers.

The benefits of using keystroke data for user authentication in online courses are many. For example, collecting keystroke data is passive and non-intrusive compared to many other authentication methods during take-home exams, which might rely on face scans [37,66] or supplying photos of yourself and a government-issued ID [20]. Additionally, some laptops and many desktop computers do not have web cameras, which are required for many other online authentication methods, whereas keyboards are used in practically all desktop and laptop computers. As keystroke data is collected passively, continuous identification is possible without a disruption to the user experience of the students in the course, while face scans or supplying a photograph require actions from the students.

One downside of keystroke-based identification is that if the data is compromised, it could be possible for an attacker to spoof keystrokes and impersonate the person whose keystrokes have been compromised [55]. This is a downside of most biometric authentication methods: similarly, if the iris of the eye is used for authentication and if the data containing information about the iris is compromised, an attacker can submit the compromised iris data as their own. This downside is similar to a password being compromised, however, with traditional passwords a user can change their password, but you cannot change your iris. It is also inconvenient to change your natural typing rhythm.

A downside from the perspective of using keystroke data for detecting if a student had someone help them in some of the assignments is that a student could have someone else complete every exercise for them, and then have that someone attend the exam for them as well. In this case, the typing profiles during the exercises and in the exam would match, raising no suspicions of cheating. This is especially a problem for courses that are offered only online, since if the exam is organized offline, students' identities can be verified in the exam.

5.3 Detecting Plagiarism from Keystroke Data

With the typing profiles, we focused on identifying students and using that information to check whether the same student attended the exam and completed the programming assignments [48, 53]. We also studied other ways of detecting plagiarism using keystroke data that are based on classifying the processes students took while programming instead of focusing on identifying the students themselves [27]. We examined whether keystroke data could be used to classify a process as either genuine (no cheating occured) or disingenuous (cheating occured). We found that there are some quite trivial ways of detecting plagiarism through keystroke data. As it is possible to see the whole process through the keystrokes, directly copy-pasting someone else's answers was very obvious. In addition to these trivial ways, we found that the linearity of the process could also indicate cheating. For example, in a genuine process students usually modify their source code quite a lot when they are refactoring their programs. Thus, a process that is very linear, where little to no refactoring occurs, can indicate plagiarism. Similar results have been later found by Yan et al. who also found that intermediate steps in the programming process can be used to detect excessive collaboration [87].

5.4 Applications Outside Education

The context of the studies included in this thesis has been computing education, more specifically learning to program. Still, the findings can have applications outside of education.

We found that students can be identified based on their typing when they are programming. It is likely that identification would also work for professional programmers who are writing source code. This finding could be used to continuously authenticate the programmer. This could be useful when working with very sensitive programs, where it is necessary that third parties cannot see the source code.

Additionally, there have been studies which have shown that emotional states can be inferred when writing natural language [21]. Since we were able to show that identification based on typing works also when writing source code, it is possible that emotions could also be inferred from keystroke data gathered from programming. Thus, automatic systems could be developed and integrated into IDEs that track the programmer's mood, and possibly suggest actions based on the programmer's mood – a short break could be suggested to a frustrated programmer.

Previous studies within the context of natural language have found that demographical information can be inferred based on keystroke data [7]. We found that students' previous programming experience and their programming performance can be inferred from keystroke data. Being able to identify someone's programming experience could be used for purposes beyond the educational context. For example, inferring the programming experience of a job candidate could be used to support hiring decisions.

Both identifying an individual and inferring their programming experience could also be used in targeted advertising. Keystroke-based identification could possibly be used to identify an individual across different websites, different browsers and computers. Even if keystroke-based identification turns out to not be viable on a large scale, just identifying some attributes of the person typing such as whether they have programmed before could be beneficial to advertisers.

5.5 Open Data and Anonymity

In Article VI, we have identified a topical issue that is relevant to the whole research community in general: there is a conflict between open data and privacy. Opening any data potentially compromises the privacy of subjects in the data. This is obviously less of an issue with certain types of data such as any data that is not related to people. For example, opening weather data is very unlikely to compromise privacy. However, opening any data related to people has the possibility of inadvertently releasing personally identifiable information [58]. Thus, researchers should be very careful when opening such data. On the other hand, especially as of late, open data has been desired by many different actors: publishing forums, funders, and the community [31]. This leaves researchers with the dilemma of how to guarantee the privacy of those who have supplied the data (in our case, students), while still trying to have the data as open as possible.

Our results in Article VI show that while it is possible to find a balance between anonymity and information, identification accuracy does not decrease linearly with increasing de-identification, and actually increases at certain higher levels of de-identification. Thus, it is not certain that "more de-identification" is better and more research on de-identification of keystroke data is needed before considering releasing such data openly. Especially since we only considered one way of identification – identification based on the timing of keystrokes. It is very likely that other ways of identification, for example based on textual content such as variable names or textual / coding style such as how one uses brackets could be viable ways of identifying the programmer. A trivial example is someone who uses their whole name in comments within the source code.

5.6 Granularity of Data

An interesting aspect of keystroke data is its fine-grainedness. The fine-grained nature of the data allows transforming it into more coarse-grained abstract data formats. For example, in our studies we have transformed fine-grained keystrokes and their timings into typing profiles. Similarly, in the context of programming, keystroke data allows the reconstruction of the whole process a programmer took to develop their program. If the state of the source code being written is only collected when, for instance, the code is compiled or when a student submits an assignment, some information related to the process is always lost.

On the other hand, there are some advantages to collecting more coarse-grained data. For example, coarse-grained data most likely requires less preprocessing compared to keystroke data, and thus analyzing it is faster. Additionally, if keystroke data is always just processed into more abstract formats for analysis, one could argue that it would be more efficient to just collect data in the abstract formats in the first place.

When considering privacy, fine-grained data could be problematic. Based on the results of our studies, people can be identified based on their typing from keystroke data. It is possible that other private or personally identifiable information could also be inferred from fine-grained data. This is an issue when keystroke data is collected for exploratory studies as new analysis methods can reveal information previously thought not to be possible to be inferred from such data. Another consideration is whether it is ethical to conduct new studies on old data post hoc. If the people who supplied the data thought that only certain types of analysis are possible or will be conducted with the data, using the data for other types of analysis could be argued to be unethical, even if individuals in the data are not explicitly identified.

5.7 Limitations

There are many limitations to both this thesis and the individual studies included. Here, we present the most important threats to both external and internal validity of this thesis. External validity relates to how well the results generalize to other contexts, and internal validity relates to how well other explanations for the results can be ruled out.

There are two main limiting factors with regards to the scope of this work. Firstly, we study keystroke data in the educational context only. Thus, the results should not be automatically assumed to generalize beyond this context. For example, it is possible that programming students write their programs differently than experts, and thus not all the results would be applicable when considering professional programmers. This brings us to the other major limiting factor, which is that most of our studies are only considering programming courses as the source of data. This was done partly due to the fact that we had keystroke data from the programming context easily available, but not similar data from other contexts. Additionally, this was also one of the main motivations behind this work – to study how keystroke data works in the programming context. On the other hand, both limiting factors regarding the context of the work – programming and education – are needed since otherwise one thesis would not be able to cover the needed studies.

All of the studies were performed within a single institution – University of Helsinki. Additionally, we only used keystroke data collected within two different courses: "Introduction to programming" and "Advanced course in programming", although we did use data from multiple iterations of those two courses. It is possible that the context of the courses affects the results.

For example, we found that students' previous programming experience and their programming proficiency can be inferred from keystroke data [49] – but it is possible that this only works for novice programmers taking their first steps in learning to program. Our courses have around the same number of students each time, which is also a limitation, since it is possible that results would be different with a different number of students. At this time, our results have not been replicated by third parties.

The analyses in this thesis are limited by the data available to us. For example, while we can recreate the programming process and infer information based on that, there are many things we do not know based on the data. If there is a long pause between keystrokes, we do not know what the student does during the pause. Additionally, the software used to collect the data relies on the difference between two subsequent states of the text, which means that we do not have information on keys that do not cause a visual change to the text, for instance the use of the arrow keys.

Many of the results in this thesis are based on building typing profiles of students. However, we cannot claim any causality. For example, even though we have found that students who are faster at typing programming-specific digraphs are more likely to perform well in the exam, it is almost certain that this is not a causal effect – practicing the speed of typing specific digraphs would presumably not increase a student's performance in the exam. A much more likely explanation would be that a person is faster at typing digraphs they have typed many times, and those who perform better have practiced programming more and thus have also typed programming-specific digraphs more.

Chapter 6

Conclusions and Future Work

In this chapter, we first revisit and answer the research questions in Section 6.1. We then present the main findings and key contributions of this work in Section 6.2. Lastly, we discuss some possible future research directions in Section 6.3.

6.1 Revisiting the Research Questions

This thesis has an overarching theme of *"What benefits are there to collecting keystroke level data in programming courses?"*. This overarching theme is studied with the following three research questions:

- RQ1. How well can students' programming performance and previous programming experience be inferred from keystroke data?
- RQ2. How can keystroke data help with detecting plagiarism and with source code authorship attribution in programming courses?
- RQ3. How could privacy and information be balanced in keystroke data?

The brief answers to these research questions, based on the publications included in this thesis, are the following:

- Answer to RQ1: Based on our results in Article I, students' programming performance and prior experience can be inferred to some extent based on keystroke data. This is done by examining how fast students type character pairs. More experienced and better performing programmers write pairs that are common in programming but rare in natural language faster.

- Answer to RQ2: Keystroke data can help with detecting plagiarism and with authorship attribution in programming courses. Firstly, in Article II we found that keystroke data can be used to construct the whole programming process character by character, and looking at the process can reveal plagiarism. When comparing students' processes, multiple students having a similar process can be a sign of excessive collaboration which is prohibited in an exam. Secondly, in Articles III and IV we found that keystroke data can be used to build individual typing profiles of the students in a course. The typing profiles can be compared, for example, between different weeks of the course, or between course exercises and exam questions, to see whether the typing profiles in different scenarios match. Having a very different typing profile in the exam compared to the exercises of a course can indicate plagiarism. Additionally, our results in Article V show that keystroke-based identification can be used to identify students writing an essay based on data collected from programming and vice versa, although the accuracy of identification suffers from the context switch.

- Answer to RQ3: As students can be identified from keystroke data, such data cannot be openly shared, at least without anonymization. An ideal situation would be such where the data could be shared while still retaining valuable information within the data. In Article VI, we found that typing profiles of students can be anonymized at least partially so that students can no longer be identified based on their typing, but their previous programming experience can still be inferred.

Based on the answers to RQs 1-3, the answer to the overarching theme is that there are many benefits to collecting keystroke data in programming courses. The benefits are mostly related to inferring information based on keystroke timings.

6.2 Key Contributions

The main finding of this thesis is that collecting keystroke data in programming courses has benefits over only collecting more coarse-grained data such as assignment submissions. Our results show that there are many attributes related to a student programmer that can be inferred from keystroke data.

Based on Article I, we have confirmed earlier results by Thomas et al. [82] who found that programming performance can be estimated based on typing. We also extended this result by showing that previous programming experience of students can be estimated similarly. The main contributions of this are the following: 1) *keystroke data can be used to estimate future performance* and thus could be used to detect students who might benefit from additional practice, and 2) when keystroke data has been collected, *students' previous programming experience can be inferred from keystrokes* in the absence of a survey asking that information (for example post hoc or for students who did not answer a survey).

Additionally, based on Article II, we have found that by reconstructing the programming process of students, possible excessive collaboration can be identified. The main contribution of this is that *keystroke data can be an additional tool to combat plagiarism in programming courses*, and could also be used to continuously monitor students' processes to intervene early when the undesired behaviour – plagiarism or excessive collaboration – has not yet become a habit.

Based on Articles III and IV, we have successfully shown that identifying someone based on typing also works when the context of the text being written is programming source code. Previous studies on the topic have mostly focused on writing natural language. The main contribution this finding has is that *keystroke data could be used in online programming courses for authorship attribution*, that is, to detect whether the same student is completing all the programming exercises. Based on Article V, this method also shows promise when identifying someone from one context to another, in our case, from programming to essay writing and vice versa. Cross-context identification based on typing is novel to the best of our knowledge, which is the main contribution of Article V.

In Article VI, we have showcased two methods for anonymizing typing profiles in a way that preserves some useful information in the typing profiles. We have shown that typing profiles can be at least partially anonymized to deter identification by keystroke analysis. The main contribution here is that *anonymized keystroke data could possibly be shared with others*, but further research is needed as we only studied preventing a single identification method.

Altogether, we have shown that keystroke data collected in programming courses has useful applications that benefit students, educators, and researchers and thus it is reasonable to collect such data.

6.3 Future Work

This thesis is a first step towards utilizing keystroke data in programming courses. There are many avenues of future research related to the findings of this thesis that could be studied. In this section, we discuss some of them.

6.3.1 Identification Based on Typing

Our studies on identification based on typing had around 200 students in the data set. Many online courses have thousands or even hundreds of thousands of students. It is likely that identification accuracy drops drastically when the number of students is increased. Thus, future work could study how to combat this problem. One possible way would be to look at the processes during exercises and the exam and based on some similarity metric determine whether the processes are similar enough instead of trying to identify students.

Additionally, in our studies identification is based on the uniqueness of the rhythm of typing, and not on for example the content or style of the text being written. Future work could study identification based on other factors present in keystroke data such as the content of the keystrokes.

6.3.2 Keystroke Data De-identification

We have shown that typing profiles can be partially anonymized while retaining valuable information. However, we only anonymized typing profiles built from keystroke data, and not the keystroke data itself. Thus, the anonymization of keystroke data that only consists of timestamped keypresses could be studied in the future. Additionally, we only focused on preventing identification that is based on the timing of keystrokes. It is likely that other identification vectors would have still allowed identification even after our anonymization. For example, identification based on the content of the keystrokes (and not the timing) could still work. Future work could explore ways to anonymize keystroke data from both timing- and content-based identification vectors.

6.3.3 Inferring Information from Keystrokes

In our studies, we have found that many attributes of students can be inferred from keystroke data. For example, their identity, their previous programming experience, and their programming performance. In the future, it would be interesting to study other possible factors that could be inferred from keystroke data. Others have found that emotional states can be inferred from keystroke data [21]. This is something that could be studied in the future in the context of programming. Inferring emotional states could help guide students towards meaningful study practices by reminding students to take breaks if their emotional state seems to require one.

6.3.4 Plagiarism

We found that the processes students take while they are programming can be used to detect plagiarism [27]. However, we analyzed students' processes post hoc, after they had completed the course and after some of the students had plagiarized their work. In the future, more methods that detect plagiarism continuously during the course – already before the exam – could be studied. If plagiarism was detected during the first occurence of it, students could be given a warning which could possibly deter further plagiarism.

Additionally, it would be interesting to study whether reasons behind copying from others could be inferred from data. For example, would it be possible to detect frustration that leads to plagiarism. It would be in the best interest of both the student and the educator to find the reasons behind plagiarism and to reduce the reasons for plagiarism instead of identifying plagiarism after it has happened.

6.3.5 Other Research Directions for Keystroke Data

All of our studies took place in programming courses. It would be interesting to study whether the results we achieved would be achievable in different contexts, such as when students are studying something else than programming. Would it be possible to infer previous experience in a course related to some other subject?

Being able to see the process a programmer takes to reach a solution could be used for many purposes. Code reviews are popular in the industry. Looking at the process a programmer took to reach a solution could be both educational to the reviewer, and also be used to self-reflect on your own programming process. Similarly, looking at how a job applicant reached a solution could tell an interviewer more than simply looking at the final state of the source code.

Another avenue for future research is to study how our results would change if the study population is changed. Many of the results of this thesis could possibly be applied in the professional context. For example, keystroke-based identification could be used with programmers working on sensitive projects to continuously monitor whether the same person is on the computer. Would we get similar results if the studies were performed with professional programmers?

References

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

[2] Alireza Ahadi, Raymond Lister, Heikki Haapala, and Arto Vihavainen. Exploring machine learning methods to automatically identify students in need of assistance. In *Proceedings of the 11th Annual International Conference on International Computing Education Research*, pages 121–130. ACM, 2015.

[3] Lauri Ahonen, Benjamin Ultan Cowley, Arto Hellas, and Kai Puolamäki. Biosignals reflect pair-dynamics in collaborative work: EDA and ECG study of pair-programming in a classroom environment. *Scientific Reports*, 8(1):3138, 2018.

[4] Kirsti Ala-Mutka. A survey of automated assessment approaches for programming assignments. *Computer Science Education*, 15(2):83–102, 2005.

[5] Ryan Baker and Kalina Yacef. The state of educational data mining in 2009: a review and future visions. *Journal of Educational Data Mining*, 1(1):3–17, 2009.

[6] Robert Bixler and Sidney D'Mello. Detecting boredom and engagement during writing with keystroke analysis, task appraisals, and stable traits. In *Proceedings of the 2013 International Conference on Intelligent User Interfaces*, pages 225–234. ACM, 2013.

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

[8] Neil Brown, Michael Kölling, Davin McCall, and Ian Utting. Blackbox: a large scale repository of novice programmers' activity. In *Proceedings of the 45th ACM Technical Symposium on Computer Science Education*, pages 223–228. ACM, 2014.

[9] Peter Brusilovsky, Stephen Edwards, Amruth Kumar, Lauri Malmi, Luciana Benotti, Duane Buck, Petri Ihantola, Rikki Prince, Teemu Sirkiä, Sergey Sosnovsky, Jaime Urquiza, Arto Vihavainen, and Michael Wollowski. Increasing adoption of smart learning content for computer science education. In *Proceedings of the Working Group Reports of the 2014 on Innovation and Technology in Computer Science Education Conference*, pages 31–57. ACM, 2014.

[10] Jane Caldwell. Clickers in the large classroom: current research and best-practice tips. *CBE – Life Sciences Education*, 6(1):9–20, 2007.

[11] Georgina Cosma and Mike Joy. Towards a definition of source-code plagiarism. *IEEE Transactions on Education*, 51(2):195–200, 2008.

[12] Coursera. Coursera signature track. `https://www.coursera.org/signature/`, 2016. Accessed: 2016-10-24.

[13] Kevin Cox and David Clark. The use of formative quizzes for deep learning. *Computers and Education*, 30(3):157–168, 1998.

[14] Tore Dalenius. Finding a needle in a haystack or identifying anonymous census records. *Journal of Official Statistics*, 2(3):329, 1986.

[15] John Daniel. Making sense of MOOCs: musings in a maze of myth, paradox and possibility. *Journal of Interactive Media in Education*, 2012(3), 2012.

[16] Jon Daries, Justin Reich, Jim Waldo, Elise Young, Jonathan Whittinghill, Daniel Thomas Seaton, Andrew Dean Ho, and Isaac Chuang. Privacy, anonymity, and big data in the social sciences. *Queue*, 12(7):30:30–30:41, 2014.

[17] Paul Denny, Andrew Luxton-Reilly, Michelle Craig, and Andrew Petersen. Improving complex task performance using a sequence of simple practice tasks. In *Proceedings of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education*, pages 4–9. ACM, 2018.

[18] Christopher Douce, David Livingstone, and James Orwell. Automatic test-based assessment of programming: a review. *Journal on Educational Resources in Computing (JERIC)*, 5(3):4, 2005.

[19] Stephen Edwards and Manuel Perez-Quinones. Web-CAT: automatically grading programming assignments. In *ACM SIGCSE Bulletin*, volume 40, pages 328–328. ACM, 2008.

[20] edX. edX verified certificate. `https://www.edx.org/verified-certificate`, 2019. Accessed: 2019-08-19.

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

[22] Hua Leong Fwa. Predicting non-completion of programming exercises using action logs and keystrokes. In *2019 International Symposium on Educational Technology (ISET)*, pages 271–275. IEEE, 2019.

[23] R Stockton Gaines, William Lisowski, S James Press, and Norman Shapiro. Authentication by keystroke timing: some preliminary results. Technical report, Rand Corp Santa Monica CA, 1980.

[24] Nasser Giacaman. Teaching by example: using analogies and live coding demonstrations to teach parallel computing concepts to undergraduate students. In *2012 IEEE 26th International Parallel and Distributed Processing Symposium Workshops & PhD Forum*, pages 1295–1298. IEEE, 2012.

[25] Philip Guo. Online python tutor: embeddable web-based program visualization for CS education. In *Proceeding of the 44th ACM Technical Symposium on Computer Science Education*, pages 579–584. ACM, 2013.

[26] Arto Hellas, Petri Ihantola, Andrew Petersen, Vangel Ajanovski, Mirela Gutica, Timo Hynninen, Antti Knutas, Juho Leinonen, Chris Messom, and Soohyun Nam 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*, pages 175–199. ACM, 2018.

[27] Arto Hellas, Juho Leinonen, and Petri Ihantola. Plagiarism in take-home exams: help-seeking, collaboration, and systematic cheating. In

Proceedings of the 2017 ACM Conference on Innovation and Technology in Computer Science Education, pages 238–243. ACM, 2017.

[28] I-Han Hsiao, Peter Brusilovsky, and Sergey Sosnovsky. Web-based parameterized questions for object-oriented programming. In *E-Learn: World Conference on E-Learning in Corporate, Government, Healthcare, and Higher Education*, pages 3728–3735. Association for the Advancement of Computing in Education (AACE), 2008.

[29] Christopher Hundhausen, Daniel Olivares, and Adam Carter. IDE-based learning analytics for computing education: a process model, critical review, and research agenda. *ACM Transactions on Computing Education (TOCE)*, 17(3):11:1–11:26, 2017.

[30] Petri Ihantola, Tuukka Ahoniemi, Ville Karavirta, and Otto Seppälä. Review of recent systems for automatic assessment of programming assignments. In *Proceedings of the 10th Koli Calling International Conference on Computing Education Research*, pages 86–93. ACM, 2010.

[31] Petri Ihantola, Arto Vihavainen, Alireza Ahadi, Matthew Butler, Jürgen Börstler, Stephen Edwards, Essi Isohanni, Ari Korhonen, Andrew Petersen, Kelly Rivers, Miguel Ángel Rubio, Judy Sheard, Bronius Skupas, Jaime Spacco, Claudia Szabo, and Daniel Toll. Educational data mining and learning analytics in programming: literature review and case studies. In *Proceedings of the 2015 ITiCSE on Working Group Reports*, pages 41–63. ACM, 2015.

[32] Kalle Ilves, Juho Leinonen, and Arto Hellas. Supporting self-regulated learning with visualizations in online learning environments. In *Proceedings of the 49th ACM Technical Symposium on Computer Science Education*, pages 257–262. ACM, 2018.

[33] Anil Jain, Lin Hong, and Sharath Pankanti. Biometric identification. *Communications of the ACM*, 43(2):90–98, 2000.

[34] Mike Joy, Georgina Cosma, Jane Yin-Kim Yau, and Jane Sinclair. Source code plagiarism – a student perspective. *IEEE Transactions on Education*, 54(1):125–132, 2010.

[35] Mike Joy and Michael Luck. Plagiarism in programming assignments. *IEEE Transactions on Education*, 42(2):129–133, 1999.

[36] Rick Joyce and Gopal Gupta. Identity authentication based on keystroke latencies. *Communications of the ACM*, 33(2):168–176, 1990.

[37] David Joyner. Congruency, adaptivity, modularity, and personalization: four experiments in teaching introduction to computing. In *Proceedings of the 4th (2017) ACM Conference on Learning @ Scale*, pages 307–310. ACM, 2017.

[38] Marcus Karnan, Muthuramalingam Akila, and Nishara Krishnaraj. Biometric personal authentication using keystroke dynamics: a review. *Applied Soft Computing*, 11(2):1565–1573, 2011.

[39] Jeffrey Karpicke and Henry Roediger III. Repeated retrieval during learning is the key to long-term retention. *Journal of Memory and Language*, 57(2):151–162, 2007.

[40] Ayaan Kazerouni, Stephen Edwards, and Clifford Shaffer. Quantifying incremental development practices and their relationship to procrastination. In *Proceedings of the 2017 ACM Conference on International Computing Education Research*, pages 191–199. ACM, 2017.

[41] Hanan Khalil and Martin Ebner. MOOCs completion rates and possible methods to improve retention – a literature review. In *EdMedia: World Conference on Educational Media and Technology*, pages 1305–1313. Association for the Advancement of Computing in Education (AACE), 2014.

[42] Agata Kołakowska. A review of emotion recognition methods based on keystroke dynamics and mouse movements. In *2013 6th International Conference on Human System Interactions (HSI)*, pages 548–555. IEEE, 2013.

[43] Agata Kołakowska. Towards detecting programmers' stress on the basis of keystroke dynamics. In *2016 Federated Conference on Computer Science and Information Systems (FedCSIS)*, pages 1621–1626. IEEE, 2016.

[44] Michael Kölling, Bruce Quig, Andrew Patterson, and John Rosenberg. The BlueJ system and its pedagogy. *Computer Science Education*, 13(4):249–268, 2003.

[45] Stefan Kulk and Bastiaan Van Loenen. Brave new open data world? *International Journal of Spatial Data Infrastructures Research*, 7:196–206, 2012.

[46] Mikko-Jussi Laakso, Tapio Salakoski, and Ari Korhonen. The feasibility of automatic assessment and feedback. In *CELDA*, pages 113–122, 2005.

[47] Juho Leinonen, Petri Ihantola, and Arto Hellas. Preventing keystroke based identification in open data sets. In *Proceedings of the 4th (2017) ACM Conference on Learning @ Scale*, pages 101–109. ACM, 2017.

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

[49] Juho Leinonen, Krista Longi, Arto Klami, and Arto Vihavainen. Automatic inference of programming performance and experience from typing patterns. In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education*, pages 132–137. ACM, 2016.

[50] Leo Leppänen, Juho Leinonen, Petri Ihantola, and Arto Hellas. Using and collecting fine-grained usage data to improve online learning materials. In *Proceedings of the 39th International Conference on Software Engineering: Software Engineering and Education Track*, pages 4–12. IEEE Press, 2017.

[51] Joshua Littenberg-Tobias and Justin Reich. Evaluating access, quality, and inverted admissions in MOOC-based blended degree pathways: a study of the MIT supply chain management MicroMasters. SocArXiv, 2018.

[52] Tharindu Liyanagunawardena, Andrew Adams, and Shirley Williams. MOOCs: a systematic study of the published literature 2008-2012. *The International Review of Research in Open and Distributed Learning*, 14(3):202–227, 2013.

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

[54] Andrew Maas, Chris Heather, Chuong Tom Do, Relly Brandman, Daphne Koller, and Andrew Ng. Offering verified credentials in massive open online courses: MOOCs and technology to advance learning and learning research. *Ubiquity*, 2014(May):2:1–2:11, 2014.

[55] John Monaco, Md Liakat Ali, and Charles Tappert. Spoofing key-press latencies with a generative keystroke dynamics model. In *2015 IEEE 7th International Conference on Biometrics Theory, Applications and Systems (BTAS)*, pages 1–8. IEEE, 2015.

[56] John Monaco, John Stewart, Sung-Hyuk Cha, and Charles Tappert. Behavioral biometric verification of student identity in online course assessment and authentication of authors in literary works. In *2013 IEEE 6th International Conference on Biometrics: Theory, Applications and Systems (BTAS)*, pages 1–8. IEEE, 2013.

[57] Fabian Monrose and Aviel Rubin. Authentication via keystroke dynamics. In *Proceedings of the 4th ACM Conference on Computer and Communications Security*, CCS '97, pages 48–56, New York, NY, USA, 1997. ACM.

[58] Arvind Narayanan and Vitaly Shmatikov. Robust de-anonymization of large datasets. In *Proceedings of the 2008 IEEE Symposium on Security and Privacy*, 2008.

[59] Hidetoshi Nonaka and Masahito Kurihara. Sensing pressure for authentication system using keystroke dynamics. *International Journal of Computational Intelligence*, 1(1):19–22, 2004.

[60] Open for business, 2017. Scientific Data Editorial. Scientific Data; 4: 170058, doi:10.1038/sdata.2017.58.

[61] Zacharoula Papamitsiou and Anastasios Economides. Learning analytics and educational data mining in practice: a systematic literature review of empirical evidence. *Journal of Educational Technology & Society*, 17(4):49–64, 2014.

[62] Andrei Papancea, Jaime Spacco, and David Hovemeyer. An open platform for managing short programming exercises. In *Proceedings of the 9th Annual International ACM Conference on International Computing Education Research*, pages 47–52. ACM, 2013.

[63] Martin Pärtel, Matti Luukkainen, Arto Vihavainen, and Thomas Vikberg. Test my code. *International Journal of Technology Enhanced Learning 2*, 5(3-4):271–283, 2013.

[64] Paulo Pisani and Ana Lorena. A systematic review on keystroke dynamics. *Journal of the Brazilian Computer Society*, 19(4):573–587, 2013.

[65] Lutz Prechelt, Guido Malpohl, and Michael Philippsen. Finding plagiarisms among a set of programs with JPlag. *Journal of Universal Computer Science*, 8(11):1016–1038, 2002.

[66] Proctortrack. `https://www.proctortrack.com/students/howitworks/`, 2019. Accessed: 2019-08-19.

[67] Teemu Rajala, Erkki Kaila, Rolf Lindén, Einari Kurvinen, Erno Lokkila, Mikko-Jussi Laakso, and Tapio Salakoski. Automatically assessed electronic exams in programming courses. In *Proceedings of the Australasian Computer Science Week Multiconference*, pages 11:1–11:8. ACM, 2016.

[68] Teemu Rajala, Mikko-Jussi Laakso, Erkki Kaila, and Tapio Salakoski. VILLE: a language-independent program visualization tool. In *Proceedings of the 7th Baltic Sea Conference on Computing Education Research – Volume 88*, pages 151–159. Australian Computer Society, Inc., 2007.

[69] Cristóbal Romero and Sebastián Ventura. Educational data mining: a review of the state of the art. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 40(6):601–618, 2010.

[70] Judy Sheard, Jason Ceddia, John Hurst, and Juhani Tuovinen. Inferring student learning behaviour from website interactions: a usage analysis. *Education and Information Technologies*, 8(3):245–266, 2003.

[71] Judy Sheard, Martin Dick, Selby Markham, Ian Macdonald, and Meaghan Walsh. Cheating and plagiarism: perceptions and practices of first year it students. In *ACM SIGCSE Bulletin*, volume 34, pages 183–187. ACM, 2002.

[72] Judy Sheard, Simon, Angela Carbone, Donald Chinn, Tony Clear, Malcolm Corney, Daryl D'Souza, Joel Fenwick, James Harland, Mikko-Jussi Laakso, and Donna Teague. How difficult are exams?: a framework for assessing the complexity of introductory programming exams. In *Proceedings of the 15th Australasian Computing Education Conference-Volume 136*, pages 145–154. Australian Computer Society, Inc., 2013.

[73] Judy Sheard, Simon, Angela Carbone, Donald Chinn, Mikko-Jussi Laakso, Tony Clear, Michael de Raadt, Daryl D'Souza, James Harland, Raymond Lister, Anne Philpott, and Geoff Warburton. Explor-

ing programming assessment instruments: a classification scheme for examination questions. In *Proceedings of the 7th International Workshop on Computing Education Research*, pages 33–38. ACM, 2011.

[74] Simon. *Emergence of computing education as a research discipline.* PhD thesis, Aalto University, 2015.

[75] Simon, Judy Sheard, Michael Morgan, Andrew Petersen, Amber Settle, and Jane Sinclair. Informing students about academic integrity in programming. In *Proceedings of the 20th Australasian Computing Education Conference*, ACE '18, pages 113–122, New York, NY, USA, 2018. ACM.

[76] Juha Sorva. *Visual program simulation in introductory programming education.* PhD thesis, Aalto University, 2012.

[77] Juha Sorva, Ville Karavirta, and Lauri Malmi. A review of generic program visualization systems for introductory programming education. *ACM Transactions on Computing Education (TOCE)*, 13(4):15, 2013.

[78] Jaime Spacco, Paul Denny, Brad Richards, David Babcock, David Hovemeyer, James Moscola, and Robert Duvall. Analyzing student work patterns using programming exercise data. In *Proceedings of the 46th ACM Technical Symposium on Computer Science Education*, pages 18–23. ACM, 2015.

[79] Jaime Spacco, David Hovemeyer, William Pugh, Fawzi Emad, Jeffrey Hollingsworth, and Nelson Padua-Perez. Experiences with Marmoset: designing and using an advanced submission and testing system for programming courses. In *ACM SIGCSE Bulletin*, volume 38, pages 13–17. ACM, 2006.

[80] Latanya Sweeney. Simple demographics often identify people uniquely. *Health (San Francisco)*, 671:1–34, 2000.

[81] Claudia Szabo, Nickolas Falkner, Antti Knutas, and Mohsen Dorodchi. Understanding the effects of lecturer intervention on computer science student behaviour. In *Proceedings of the 2017 ITiCSE Conference on Working Group Reports*, pages 105–124. ACM, 2018.

[82] Richard Thomas, Amela Karahasanovic, and Gregor Kennedy. An investigation into keystroke latency metrics as an indicator of programming performance. In *Proceedings of the 7th Australasian Confer-*

ence on Computing Education – Volume 42, pages 127–134. Australian Computer Society, Inc., 2005.

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

[84] Arto Vihavainen, Matti Luukkainen, and Jaakko Kurhila. MOOC as semester-long entrance exam. In *Proceedings of the 14th Annual ACM SIGITE Conference on Information Technology Education*, pages 177–182. ACM, 2013.

[85] Arto Vihavainen, Matti Paksula, and Matti Luukkainen. Extreme apprenticeship method in teaching programming for beginners. In *Proceedings of the 42nd ACM Technical Symposium on Computer Science Education*, pages 93–98. ACM, 2011.

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

[87] Lisa Yan, Nick McKeown, Mehran Sahami, and Chris Piech. TMOSS: Using intermediate assignment work to understand excessive collaboration in large classes. In *Proceedings of the 49th ACM Technical Symposium on Computer Science Education*, pages 110–115. ACM, 2018.

TIETOJENKÄSITTELYTIETEEN OSASTO
PL 68 (Pietari Kalmin katu 5)
00014 Helsingin yliopisto

DEPARTMENT OF COMPUTER SCIENCE
P.O. Box 68 (Pietari Kalmin katu 5)
FI-00014 University of Helsinki, FINLAND

JULKAISUSARJA **A**

SERIES OF PUBLICATIONS **A**