Educational Data Mining and Learning Analytics in Programming: Literature Review and Case Studies

Petri Ihantola[*]
Tampere Univ. of Technology
Tampere, Finland
petri.ihantola@tut.fi

Arto Vihavainen[*]
University of Helsinki
Helsinki, Finland
avihavai@cs.helsinki.fi

Alireza Ahadi
Univ. of Technology, Sydney
Australia
Alireza.Ahadi@uts.edu.au

Matthew Butler
Monash University
Melbourne, Australia
matthew.butler@monash.edu

Jürgen Börstler
Blekinge Instit. of Technology
Sweden
jubo@acm.org

Stephen H. Edwards
Virginia Tech
Blacksburg, VA, USA
edwards@cs.vt.edu

Essi Isohanni
Tampere Univ. of Technology
Tampere, Finland
essi.isohanni@tut.fi

Ari Korhonen
Aalto University
Finland
ari.korhonen@aalto.fi

Andrew Petersen
Univ. of Toronto Mississauga
Mississauga, Canada
petersen@cs.toronto.edu

Kelly Rivers
Carnegie Mellon University
Pittsburgh, PA, USA
krivers@cs.cmu.edu

Miguel Ángel Rubio
University of Granada
Granada, Spain
marubio@ugr.es

Judy Sheard
Monash University
Australia
judy.sheard@monash.edu

Bronius Skupas
Vilnius University
Lithuania
bronius@gmail.com

Jaime Spacco
Knox College
Galesburg, IL, USA
jspacco@knox.edu

Claudia Szabo
The University of Adelaide
Australia
claudia.szabo@adelaide.edu.au

Daniel Toll
Linnaeus University
Sweden
daniel.toll@lnu.se

ABSTRACT

Educational data mining and learning analytics promise better understanding of student behavior and knowledge, as well as new information on the tacit factors that contribute to student actions. This knowledge can be used to inform decisions related to course and tool design and pedagogy, and to further engage students and guide those at risk of failure. This working group report provides an overview of the body of knowledge regarding the use of educational data mining and learning analytics focused on the teaching and learning of programming. In a literature survey on mining students' programming processes for 2005–2015, we observe a significant increase in work related to the field. However, the majority of the studies focus on simplistic metric analysis and are conducted within a single institution and a single course. This indicates the existence of further avenues of research and a critical need for validation and replication to better understand the various contributing factors and the reasons why certain results occur. We introduce a novel taxonomy to analyse replicating studies and discuss the importance of replicating and reproducing previous work. We describe what is the state of the art in collecting and sharing programming data. To better understand the challenges involved in replicating or reproducing existing studies, we report our experiences from three case studies using programming data. Finally, we present a discussion of future directions for the education and research community.

CCS Concepts

•**General and reference** → **Surveys and overviews;** •**Social and professional topics** → **Computing education;** •**Applied computing** → **Education;** •**Information systems** → *Decision support systems; Data mining; Users and interactive retrieval;* •**Security and privacy** → Human and societal aspects of security and privacy;

Keywords

educational data mining; learning analytics; programming; replication; literature review

[*] Working group leaders

ITiCSE WGR'16, July 4–8, 2015, Vilnius, Lithuania.

ACM ISBN 978-1-4503-4146-2/15/07..

DOI: http://dx.doi.org/10.1145/2858796.2858798

1. INTRODUCTION

Understanding why students succeed or fail in computer science courses is a fundamental drive for computer science education research (CSEd), yet, as a research community, we are still struggling to routinely collect data to systematically study and explore this as identified by Lister in 2010 [75]. This is despite initial promising works at the beginning of the century [43, 59, 83, 116]. With the use of online learning systems has come a significant growth in the capture and analysis of student performance data. This focus on systematic collection and analysis of learner-oriented data is also based in part on the drive to move from more subjective, anecdotally-oriented CSEd experiences, to empirically-based, data-driven research methods. This strategy can lead to more rigorous, higher-quality evaluation of proposed teaching methods or interventions, as well as help in teasing out the tacit factors that contribute to the observed learning outcomes.

This ITiCSE 2015 working group focused on creating a foundation for CSEd research based on mining data collected as students work on programming problems. Literature suggests that there has been a substantive rise in data collection for evidence-based research in programming. Little, however, has been done to date on relating studies to each other, categorizing the types of data and the ways in which it is collected, or reproducing findings with similar data sets. As such, the main objectives of this work have been to: (i) systematically identify and analyse relevant works in the literature; (ii) meaningfully categorize these studies; and (iii) identify discipline-specific challenges related to replication and reproduction.

To address the above objectives, we identified and surveyed existing literature in this area to gain an understanding of the different approaches being used. We analyzed the critical commonalities and differences in the tools and strategies being used to collect and analyze data from student work processes and artefacts, and examined issues and challenges related to reproducing prior studies and results. This emphasizes the value in relating studies to each other and encourage the reproduction—the deliberate re-investigation of a question in a similar (but not necessarily identical) context—of published studies. To better understand the landscape of replication studies, we propose a taxonomy that classifies existing works based on how three main criteria, namely researchers, analysis methods, and production, change with respect to the baseline study.

Our vision for routine data collection as students work on classroom activities goes far beyond providing a solid basis for evaluating classroom research questions. Once researchers become accustomed to systematically collecting learner data to support research, these methods will eventually grow to see use by practitioners as well. In the future, by routinely collecting performance and skill data as students work in a course, data analytics and support tools can be regularly applied to provide instructors with greater insight into what is actually occurring in the classroom, opening up new opportunities for identifying individual student needs, providing targeted activities to students at the ends of the learner spectrum, and personalizing the learning process. This research aims to support moving toward a data-driven practice of teaching that uses the best ideas from continuous process improvement in a learning context.

To provide a head start to the process of replication, we introduce a novel taxonomy that can help other researchers to identify different types of replication studies, which provides an overview on what types of replications can be done in general. This and challenges of replication in general are discussed in Section 3. Section 4 describes how programming data is collected, lists important attributes of the data sets being used in current research and advocates for more standardization in the reporting of meta-data and the use of data collection infrastructures. Section 5 focuses on the issue of reproducibility and includes three case studies that illustrate the challenges encountered when reproducing existing studies. Finally, the article concludes with a discussion of the grand challenges facing the field in Sections 6 and 7, respectively.

2. LITERATURE SURVEY

The driving questions for the literature survey were the following:

- What information about the teaching and learning of programming can be gained through the analysis of programming data?
- Which of that data can be collected and analyzed automatically?
- What are the challenges of using the study results in practice?

2.1 Method

For the survey, we adopted a light version of the guidelines for systematic literature reviews [70].

2.1.1 *Identification of Relevant Literature*

Constructing an effective search string for literature dealing with "programming data" that can be "analyzed automatically" proved infeasible. To search for literature, the group started out with a set of relevant papers that should be found with a database search (Gold Standard). Pilot searches on several databases[1] with varying search strings returned too many irrelevant papers and few papers from our Gold Standard. Even restricting the domain to the teaching and learning of programming returned too many irrelevant publications, and missed many relevant papers. The group could not define a search string that resulted in an acceptable balance between precision (result included only relevant papers) and recall (result included all relevant papers).

To solve this problem, we identified a number of key venues (see Table 1) that were searched manually. To ensure the reliability of the search process, two independent reviewers were allocated to each venue to search through the years 2010–2015 and include/exclude papers based on their titles and abstracts. A description of the inclusion/exclusion criteria can be found in Section 2.1.2. Since the inter-rated agreement was almost 100%, the remaining years (2005–2009) were searched by only one reviewer. To avoid a premature exclusion of potentially relevant papers, we adopted an inclusive approach; all undecided papers were included in a first step.

All included and undecided papers were then compiled into a "master list" and checked by a second reviewer. After this step, the master list contained 101 papers that were subject to data extraction and quality assessment.

[1] We used the ACM and IEEE Digital Libraries, Scopus, Springer Link and Google Scholar.

Table 1: Venues searched for relevant articles, years of exploration, total articles explored, final articles included after inclusion/exclusion criteria, data extraction, and quality assessment.

Venue	Years	Number of articles	Articles included
ACE – Australasian Computing Education Conference (ACM)	2005–2015	240	6
CSE – Computer Science Education (Taylor&Francis)	2005–2015*	168	4
EDM – International Conference on Educational Data Mining	2008–2015	327	6
ICER – International Computing Education Research Workshop	2005–2014	162	8
ITiCSE – ACM Annual Conference on Innovation and Technology in Computer Science Education	2005–2015	694	20
JEDM – Journal of Educational Data Mining	2009–2015*	34	2
Koli Calling – Conference on Computing Education Research	2005–2014	202	8
PPIG – Psychology of Programming Interest Group Annual Workshop	2005–2014	166	1
TOCE/JERIC – ACM Transactions on Computing Education / ACM Journal of Educational Resources in Computing	2005–2015*	195	1
SIGCSE – The ACM Technical Symposium on Computer Science Education	2005–2015	1162	20
TLT – IEEE Transactions on Learning Technologies	2010–2015*	221	0
Total		*3571*	*76*

*January–June for 2015.

2.1.2 Inclusion/Exclusion Criteria

The inclusion of an article was determined based on the following criteria:

- *The programming problems are open-ended*—We excluded systems that are custom-defined for certain types of problems or algorithms as they are not easily transferable to other contexts. Regarding the programming language, we included only languages or systems in which programs are composed of statements and control structures in the form of text or textual blocks. This includes all general-purpose programming languages, but also systems like Scratch, where programs are composed of blocks that contain such elements. Visual languages or systems, such as Kara [51] for example, were excluded.

- *Data collection relates to the programming process*—The data should make it possible to analyze the approaches that a student is taking while solving a problem and their way of working. This can include sequences of assessments or coding snapshots, meaning that related multiple data points are collected, as well as single data points collected a number of times, e.g. a summary of compilation results for a class.

- *Data collection and analysis is automated*—This is essential for facilitating real-time support and in contexts where human support is limited or restricted. This is particularly critical in situations where traditional supervision does not scale, since students are distributed and number in the thousands, like in many MOOCs and institutions with very large classes.

- *Length > 3 pages*—Short papers were excluded.

2.1.3 Data Extraction and Quality Assessment

For data extraction, we developed a form based on the driving questions for the literature survey, described above, and Malmi et al.'s categorization of computing education research [79]. Quality assessment was done together with data extraction. For this purpose, we adapted the quality criteria for research proposed by Downs & Black [34] and Runeson & Höst [95] and added them to the data extraction form.

The final data extraction form contained the following main categories:

- Paper identification
- Methodological aspects of the research
- Context of the research, including details about the course, the subjects, and the tasks the subjects had to carry out.
- Data that was collected and how it was collected and analyzed
- Overall results
- Quality assessment

All items of the data extraction sheet can be found in Appendix A. Of the 101 articles that were left after the inclusion/exclusion phase another 25 were excluded during the data extraction phase. The full list of primary papers can be found in Appendix B.

2.2 Results

We analyzed 76 papers distributed over the time period from 2005 to 2015, as shown in Figure 1. As it can be seen, the rate of papers being published on this topic has increased rapidly over the past ten years. To determine whether this was due to increased interest in the subject or whether it was just due to more papers being published in the searched venues, we also analyzed the number of papers greater than 3 pages long published in each venue per year. We found that almost all of the chosen venues had a constant rate of publication, the only exception being the International Conference on Educational Data Mining, which has grown rapidly over the past eight years (from 17 papers in 2008 to 91 papers in 2015). This implies that there is a general increase in interest in educational data mining, in addition to the subfield of educational data mining of programming data.

The papers which met the inclusion criteria came from almost all of the venues we investigated (see Table 1). The two largest conferences, SIGCSE and ITiCSE, accounted for 40 (53%) of the included papers. A large number of included papers were also published at the ICER conference (8/162). In contrast, we found no papers that could be included from TLT, and very few from the ACM Transactions on Computing Education, previously known as JERIC (1/195), and the Psychology of Programming Interest Group Workshop PPIG (1/166).

Our analysis will next explore the results of the data extraction, starting from a thematic analysis of the research goals, and finishing with the assessment of quality.

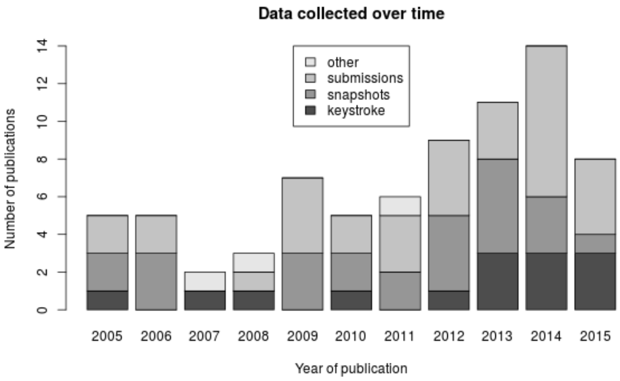

Figure 1: The number of included papers published in each of the specified years. The different colors indicate different levels of data collection, from low-level (keystrokes) to high-level (submissions). Data from 2015 includes only publications between January and June.

2.2.1 Research Goals

The research goal and motivation were extracted from each article and subsequently analyzed by three researchers, who performed a thematic analysis. The researchers first labeled the extracts individually, after which the labels were discussed jointly to form a common consensus on the main themes and goals. During the process, three overall categories emerged: *student*, *programming*, and *learning environment*. Articles that were placed within the *student* category were concerned with, for example, predicting student performance, student affect and estimating students' knowledge. Articles in the *programming* category were concerned with for example, identifying programming behaviour and strategies and errors made in programming, while the *learning environment* category was related to (students' use of) tools and automated testing, grading and feedback, and the like.

For each category, a number of subcategories were identified. As is natural for such analysis, there were cases where the original authors did not refer to their research goals using the same terminology that emerged during the thematic analysis – in some cases, the terminology of the article referred to other subcategories. Such cases were resolved by the researchers after careful analysis of each paper, and in some cases, if a single category could not be agreed upon, the article was placed in more than a single category. Such cases were visible in works that focused on common errors from the programming process, but also correlated those errors with course outcomes, categorized under the *programming* category, while work that focused on course outcomes and students' performance, but also analyzed students' programming errors and correlated those with course outcomes, was categorized under the *student* category. The results of the categorization are given in Table 2.

2.2.2 Approach

The "Approach" classification sought to identify in the papers the nature and breadth of the research undertaken. The type of research was classified into one of six categories: case study; constructive research; experiment; study; survey research; or other. Multiple categories could be chosen, for example both constructive research and study may be chosen if the paper reported in depth on the development of a tool and presented data on its use in a classroom setting. Of the papers, 59 (78%) were considered as a study, where results reported data collected in a natural setting, such as in class activity. This is in contrast to only 11 (14%) that were classified as experimental research, where formalised experimental environments were set up in order to collect the data. This suggests that researchers prefer to collect programming process data in situ, possibly hoping to understand student programming behaviours as they occur in more natural environments. It is also important to highlight that 15 (20%) were categorised as constructive research, suggesting there is a perceived importance placed on reporting in detail the construction of tools to aid in the collection of programming process data.

Very few studies were founded on formal theories or known pedagogical models. While many papers would mention some underpinning educational theories, only 8 (11%) formally referenced and utilised a specific theory or model in the development of a tool and/or the analysis of the data collected.

When considering the breadth of the research and data presented, of key interest is that the majority of studies (62 studies, 81%) presented work conducted within just a single institution. Reasons for the lack of multi-institutional work (in particular data collection) are unclear. One perspective may be the difficulties, or perceived difficulties, with data privacy and ethical issues. However, as will be discussed further in a later section, very few papers raised any issues regarding the ethics or privacy of their work or data.

Few of the studies suggested any breadth of work in the form of longitudinal data collection or data collected across multiple courses. Only 15 (20%) studies presented work that could be considered longitudinal in nature, where data was collected from students over multiple offerings of a course or courses. More positively, 26 studies (34%) presented data captured across multiple courses. This however is still low, and while proof of concept studies are important, it highlights that for rigour of work in the area, more studies must be undertaken that seek to capture data from more than just a single offering of a course.

Given the nature of the type and breadth of the data presented in the literature, it is unsurprising that there were very few replication or reproduction studies represented. Indeed, only 5 (7%) were studies that sought to replicate or reproduce previously published work. Reasons for this low result are unclear, however may relate to the infancy of work in this area or inherent difficulties in building upon existing work, whether this be for technical reasons or those relating to privacy and ethics.

2.2.3 Context

Context sought to understand the programming language and environment being used for process data collection. The vast majority of studies focused on programming within a course context (61 studies, 80%). Ten studies (13%) indi-

Table 2: Categorization of the research goals

Category	Short Description or Example	Articles
Students'		
Ability and knowledge	Approaches for evaluating students' knowledge of specific concepts	[13, 67, 117]
Affective states	Approaches for evaluating students' affective states during the programming process	[78]
Behavior	Students' behaviors specific to a system as well as other activities	[46, 50, 88, 89, 97, 101, 102]
Difficulties	Difficulties and e.g. concepts that are challenging for students	[25, 27, 58, 64]
Drop-out risk and performance	Approaches for identifying students that are at risk of dropping out from class, as well as measuring performance	[45, 61, 66, 71, 77, 104, 111, 112]
Environment		
Algorithm analysis	Analysis of student constructed programs and algorithms for e.g. automatic categorization of students' solutions	[92, 105, 106]
(Automated) feedback	Improving and estimating feedback mechanisms	[11, 12, 24, 29, 33, 44, 93]
Automated grading	Analysis of solutions for e.g. reporting and grading	[9, 55]
IDE usage	Analysis of students interactions within the programming environment	[8, 37, 80, 113, 115]
Testing	Approaches for improving automated testing of students' source code	[41, 42, 99]
Programming		
Behavior	Collection of coding, compiling, debugging, and testing activities and their associated metrics that students perform	[43, 56, 60, 60, 61, 74, 99, 111]
Errors	Work related to understanding errors during the programming process, e.g. syntax errors	[5, 6, 10, 17, 30–32, 36, 49, 82, 98]
Patterns	Repeated sequences of events within a programming behavior	[83, 91]
Process	Programming behavior in which activities follow a sequence	[52, 53]
Progress	Estimating whether the solution is approaching a goal	[7, 104]
Strategies	Approaches to design of a solution and associated programming behavior	[20, 22, 23, 68, 69, 96, 103, 109, 114, 115]
Metrics	Focused on metrics, did not necessarily attempt to gain an understanding of programming behavior, process, or strategy	[18, 84, 107]
Testing Behavior	Analysis of students' source code testing behaviors	[21, 40]

cated that process data collection was undertaken outside a formal course context, while the remaining 5 studies did not explicitly state the research environment.

The study course environment was predominantly introductory programming courses (30 studies, 39%). This captured those explicitly named as such, as well as those often labelled as CS1. Follow on, or more advanced courses such as CS2, were captured in 11 (14%) of the studies reviewed. Miscellaneous university-level courses such as Data Structures and Algorithms, Object-Oriented Design, Web Programming, and variations accounted for the rest. Three studies indicated that process data had been collected in pre-CS1 or "after school" contexts. One key statistic of note is that only 3 studies explicitly mentioned that more than one course was studied, again highlighting the lack of breadth in the studies examined. Unknown courses accounted for 26 studies (34%). Given that most studies reviewed were single course/single institution, it was curious that many studies did not report the course context of their study, however as indicated above, 10 studies did acknowledge that data was collected outside a course context.

Regarding the programming language, Java was the most common programming language being studied (45 studies, 48%). Python was considerably less with only 8 instances (11%), and C++ with 5 (7%). Of the papers surveyed, 13 did not explicitly state the language being used. Of note is that only 7 (9%) mentioned that more than one language was examined. The reasons for the dominance of Java need to be examined further. It was not clear from the studies if this was based on characteristics of the language specifically, characteristics of available development environments, or simply what was used in the course in which students were enrolled.

Integrated Development Environments were the most varied context element. Given the prevalence of Java, it is unsurprising that BlueJ (14 studies, 18%) and Eclipse (6 studies, 8%) were two of the more widely mentioned. From all the studies, 23 (30%) did not acknowledge a specific development environment.

2.2.4 *Subjects*

The nature of the participants was captured by looking at how many students were part of the data collection process and at what level of study they were. The minimum number of students recorded was 10, however one study reported having collected data from 265,000 students (although these may not be different students). The spread of numbers was fairly even. 17 studies (22%) reported having less than 100 students from which data was gathered. The largest proportion of studies had between 100 and 500 students involved (32 studies, 42%). Only 2 studies (3%) had between 500 and 1000 student participants, however 12 studies (16%) reported having process data from over 1000 students. It is encouraging to see that so many studies had such large data sets, as it makes it easier to apply more traditional educational data mining methods. 13 studies did not report specific numbers.

Of the study level of the students, it was often difficult to ascertain the exact nature of the students, in part due to differing terminology, but in many cases due to reports not specifying what specific level of study students were at. The majority of students were specified to be first year or CS1 students (31 studies, 41%), while 12 studies (16%) specified that students were at a second year or CS2 level. At a more advanced level, 2 studies (3%) reported students being in their 3rd year of study, and the same number reporting data collected from postgraduate students. It is important to note that 7 studies (9%) indicated that the nature of the students was mixed, in most cases still at a university level. Of other note is that in 8 studies (11%), the students were at a High School or pre-CS1 level. In 14 studies (18%) the specific level of the students could not be determined.

2.2.5 Task

The Task classification sought to capture the nature of the programming task/s that students were asked to undertake in order to capture the process data. Most studies (61 studies, 80%) required students to complete multiple programming tasks, rather than just a single task. Specific detail on the task/s was often difficult to ascertain from the literature. Many studies described them simply as "small programming tasks", "programming problems" or "programming assignments". In some cases specific detail was provided describing the nature of the task. Examples of the programming tasks being assigned to students included implementing specified algorithms (such as searching and sorting), creating graphics, and even implementing computer games.

What could be gleaned from the papers were some fundamental characteristics of the tasks. To begin, all were described (or could be recognised to be) open ended tasks. This was an inclusion criteria of the selected literature and the categorisation process confirmed that this was the case. In 47 studies (62%) it was explicitly stated that the tasks were being assessed, with only four studies (5%) indicating that the tasks were not. The presence of assessment could not be determined in the remaining 25 studies reviewed. It is surprising that so many studies either focused on, or required that, the programming tasks being undertaken were assessable. However it is also likely that many of those that were undetermined may not have been being assessed. Finally, 37 of the studies (49%) stated explicitly that the data collection process also afforded automated feedback back to the students. Of the other studies, 13 (17%) did not provide such feedback, and the remaining 26 did not explicitly state one way or the other.

2.2.6 Data and Collection

Out of 76 studies, 24 (32%) stated that they informed students that data would be analyzed, and only 6 studies (8%) specifically stated that the data collected was anonymized before analysis. This is not to say that the remaining papers did not inform students or anonymize the data; they simply did not report what had been done in the paper. In terms of the level of data reported, we found that papers trended towards logging high-level data (submissions, $n = 33$) more often than low-level data (keystrokes, $n = 14$). Comparing the level of data collected to the year in which the paper was published revealed that keystroke-level data has only been regularly used for data collection in the last few years.

Additionally, we found that open data sets are exceedingly rare in the literature we reviewed. Only 4 studies (5%) were based on open datasets. Three of these studies were using the Blackbox dataset [18], while the last provided open tools and detailed data within the paper.

As the purpose of the studies was to examine students' programming processes, a straightforward way to set up a data collection could be to use the program solutions submitted by students during a programming course. To obtain more fine-grained information about students' actions, it is also possible to automatically log students' actions as they work on the programming assignments. The log information can, for example, contain all the interactions the student completes within the IDE or even the individual key strokes.

In 47 of the studies (62%) the automated data collection was conducted by instrumenting the programming environment or the computer that the students used so that the programming environment automatically logs students' actions for research purposes. In 21 studies (28%) there was no instrumentation of the programming environment. In these cases the research data consisted of materials that the students explicitly submitted. Eight of the studies (10%) did not clearly state if the programming environment was instrumented for data collection. The most often mentioned data collection instruments were WebCAT (nine studies), BlueJ or an extension of BlueJ (eight studies), and CodeWrite or its plugin (five studies).

To complement the automatically collected information, other data collection was also included in the studies. The most typical data collection method was the use of a questionnaire (16 studies out of 76, 21%). 8 studies (11%) used manual assessment outcomes, for example assignment grades, exam results or final marks of the students. Observations or interviews were included in 3 studies (4%). One study used screen recordings that were coded qualitatively, statistics of video views and students' actions on discussion forums.

2.2.7 Analysis Methods

Regarding results analysis, a variety of analysis methods were conducted. Descriptive statistical methods were most common (63 studies, 83%), with most studies at least reporting basic counts and percentages. 22 studies (29%) also conducted more detailed statistical analysis, such as inferential, Bayesian, t-test, and the like. 14 studies (18%) conducted some form of exploratory statistical analysis, such as correlation, regression, or factor analysis.

Methods of interpretive analysis were also present in the studies reviewed. Interpretative classification, such as classifying based on an existing classification scheme or one that is refined during the analysis, was present in 6 studies (8%). Interpretive qualitative analysis, such as the data-driven formation of qualitatively different categories, appeared in 12 (16%) papers. Automated classification was accounted for in 11 studies reviewed (14%). Finally, 3 papers did not present any formal analysis of data at all.

2.2.8 Quality

We performed a quality assessment of the reviewed papers by selecting 15 questions to evaluate each paper on. The questions which were used to assess the quality of the reviewed papers were adapted from quality criteria for research by Downs & Black [34] and Runeson & Höst [95]. The final quality criteria questionnaire is a part of the Data Extraction Form included as Appendix A.

We found a normal distribution of quality scores across studies, with an average score of 7.37/15 (sd 3.787). In examining individual assessment questions, we found several items which the group of papers did particularly well on, and a number which were sorely lacking. We describe the items here, with suggestions for future work.

The most-often-met quality metrics were "The hypotheses/aims/objectives of the study are clearly described" (with 79% of the papers meeting the metric) and "The main findings of the study are clearly described" (also 79%). This shows that the majority of papers had clear goals, were successful at demonstrating what they had achieved, and adequately shared their findings.

On the other hand, the least-often-met quality metrics include "Confounding factors have been acknowledged and discussed" (25%), "Threats to validity are analysed in a systematic way and countermeasures were taken to reduce threats" (22%), and "Ethical issues were acknowledged and discussed" (3%). The first two items demonstrate that the studies do not often consider factors outside of the collected data, despite the fact that most studies did not collect or present demographic information (which could lead to many confounding factors). The last item demonstrates that the field has not yet engaged in a general discussion of the issues related to privacy and ethics that other branches of data mining have dealt with.

2.3 Concluding Remarks

Overall, during the literature review, we observed that there has been a significant increase in the amount of articles that are exploring student-constructed solutions to programming problems. The underlying themes were often related to approaches for helping the student or the teacher, such as evaluating approaches for providing feedback, identifying at-risk students, or extracting programming strategies or patterns that could, perhaps, be used during the instruction to inform students on their choices.

A number of papers were also driven by a post-hoc analysis approach, such as course results and their connection to variables extracted from the context. Whilst such studies are valuable to the field, this also provides evidence of the relative immaturity of the field, as longitudinal studies, where the effect of an intervention would be measured across a number of courses or institutions, are virtually non-existent.

The lack of cross-institution co-operation may be explained by ethics and privacy issues related to working with student data, which were not often discussed in the papers. At the same time, similar to the working group on Smart Learning Content at ITiCSE'14 [19], we also observed the existence of a number of tools and systems used for collecting process data, and that those tools are not often shared or used across institutions.

The relative immaturity of the field is also evident in the way in which confounding factors are reported, and how often researchers sought to replicate or reproduce previous work. Whilst novelty is valued, building a more coherent picture of the factors that contribute to the observations needs to take place for the field to evolve.

3. VERIFYING EXPERIMENTAL FINDINGS

3.1 Re-analysis, Replication, and Reproduction

As discussed above, much of the research in our field is isolated and attempts at verifying the results and findings of others are rare. Juristo et al. [47] studied how several disciplines verify experimental findings. They identified three major groups of methods. Based on this analysis, they proposed to use the terms re-analysis, replication, and reproduction for identifying verification methods.

In **re-analysis**, *data* from a previous experiment is used to verify the results, i.e. there is no change in the production. Re-analysis can be used to verify that no errors were made during the data analysis phase and that similar findings can be obtained using the same data as in a previous experiment.

In **replication**, the *method* from a previous experiment is followed to verify the results. Replication can be used to verify that the observed findings can be discovered more than once with the same method. For example, in software related studies the experimenter can vary the subjects (students) or the software artifacts (tools, systems).

In **reproduction**, the *hypotheses* from a previous experiment are tested to verify that the findings are independent of the experimental method used. This ensures that testing the hypothesis is not dependent on any particular procedure, materials or instruments used in the experiment.

Based on the literature review, replication, i.e. changing subjects and/or used tools, and reproduction, i.e. changing the data analysis approaches and methods but seeking the same phenomenon, as well as their combination, were the most common ways of verifying previously observed findings in our field.

3.2 The R.A.P Taxonomy

Our literature review showed that reproduction studies can be performed by the same or different researchers, can create new data, or involve different types of analyses or key considerations, while still looking at the same dataset as the baseline study. To better understand the landscape of reproduction studies, we propose three main criteria, namely, *researchers*, *data analysis*, and *production*. The researchers criterion captures whether the same group of researchers is involved in the reproduction and baseline studies. The data analysis criterion captures whether the same or new data analyses were performed, whereas the production criterion captures whether new data or information is produced.

An extreme interpretation of reproduction is that all three key components, e.g. data, researchers, and the method used to analyze the data set, should be renewed or designed anew. However, studies in which all three components are not necessarily changed do exist, and in fact, any of the three can be changed or kept intact to produce results of interest. This is illustrated in the Venn diagram in Figure 2. In it, the three components of reproduction are intersected to highlight the different types of studies that can be undertaken, and to provide insight into the specific significance of the work being undertaken. The most strict interpretation of reproduction, where all three research elements are changed, is depicted in the middle as 'R.A.P'. At the other extreme, if none of the components or only researchers (R) are changing, this signifies that researchers are simply re-analyzing or reviewing previous data or studies. In the following, we systematically go through all possible combinations in order to highlight the significant contributions each combination can make.

The baseline for the naming conventions used are derived from [47] in which the authors analyzed 18 replication classifications. Based on those, they identified the three major groups that were described above. To distinguish between these categories and to elaborate the notion of reproduction in more detail, we propose a more detailed novel classification to categorize different types of studies. Thus, we have introduced additional labels in order to recognize the difference among these. In the following, the two extremes highlighted earlier (re-analysis and reproduction) still exist. However, we can see that there are a number of replication studies that are different from each other that lie within these two extremes.

- R = re-analysis—a different experimenter is following the same analysis done before with the original data set for review purposes.
- A = extended analysis—an experimenter is extending the baseline study by looking at a previously analyzed data set, but using new analysis methods.
- P = repetition—an experimenter is repeating the same analysis with new data, e.g. applying proven analysis techniques on a different data set, or collecting similar data each year in order to undertake a longitudinal study.
- R.A = verification—the same data set is looked into again by a different experimenter and using a different analysis method to verify the conclusions.
- R.P = replication study—a different experimenter is following the same analysis method as in the baseline study, but using their own different data set.
- A.P = triangulation—an experimenter is collecting a new data set to be analyzed with a new method.
- R.A.P = reproduction—a different experimenter is analyzing their own new data set and following a new analysis method designed for the study in order to test the hypotheses in the baseline study.

We understand that the strict requirements of reproduction emerge from natural sciences – where a new finding should be verified by many different research groups in different places until a hypothesis is accepted by the community. However, in our discipline, many other types of verification studies can be designed and still make a significant contribution to the field. The taxonomical approach described here can help guide other researchers in identifying the ways in which their work contributes to the existing body of knowledge.

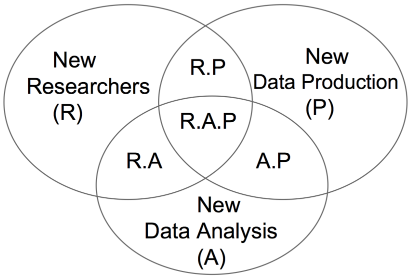

Figure 2: Venn diagram of various study types supporting verification of previous research. Each verification study can be classified by determining whether one, two or all three of the main components—Researchers, Analysis method, and/or Production—are changed compared with the baseline study.

4. DATA COLLECTION

Data are the fuel of educational data mining and learning analytics. Collecting and sharing data plays a key role in verifying experimental findings. Without data sharing, verification and re-analysis is simply not possible. In order to replicate or reproduce a previous study, one needs to collect data that is related to the previous research. Because of the significance of the topic, we describe the state of the art in data collection of programming traces as well as the differences in various data collection approaches, including the granularity of the data which describes how students have solved programming problems. Finally, we also consider ethics related to gathering and sharing such data.

4.1 How Data are Collected

If researchers wish for data collection on student behaviors to be routine and systematic, the only practical approach is to automate. Different researchers have taken different approaches to capturing or recording interactions between learners and programming environments [1–3, 11, 14, 18, 26, 31, 36, 38, 43, 48, 54, 57, 62, 63, 65, 72–74, 83, 86, 88, 93, 96, 100, 103, 113, 118]. First, we can roughly describe the differences in terms of *granularity*—approximately referring to the size or frequency of the events, or how frequently the state of the solution is captured [110]. Figure 3 depicts the most common points on the granularity spectrum from smallest (individual key strokes) to largest (complete assignment submissions to some form of assessment or feedback system). In many cases, differences between the data collected and used by different researchers can be characterized by describing which point(s) on the granularity continuum were chosen, and whether event actions, event actions plus feedback, or the states of the solution were captured.

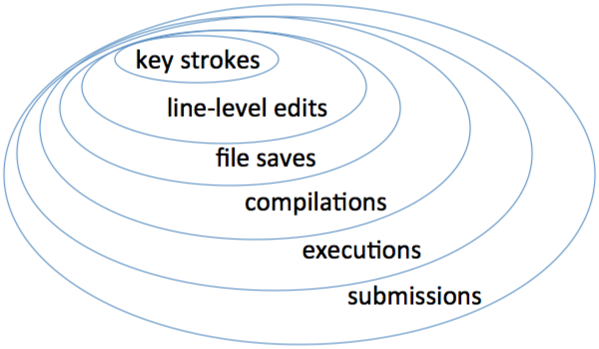

Figure 3: Data can be collected at different levels of granularity, which implies different collection frequencies and associated data set sizes.

Among the common tools used for data collection, differences emerge in the same way. Researchers have commonly used different varieties of tools for systematic collection of student data:

- *Automated grading systems* – tools used to collect and process student work that is presented for assessment are commonly used in data collection. These systems typically result in data sets at the granularity of *submissions* (a complete solution state representation, usually identified by the student as ready for evaluation). Although full event information about the student's submission action and the associated feedback received may be recorded, a significant limitation is the relative sparseness of these events, and the lack of visibility into solution states or actions between submission events.

Web-CAT [39] is a typical example of a tool used for this kind of data collection.

- *IDE instrumentation* – tools used to collect individual events within a student's IDE usually focus on "project"-level events, including file saving, compilation, and execution. Some systems, such as HackyStat [63], focus more on recording information about events, while others, such as Marmoset [103], focus on capturing snapshots of the solution being constructed. Some web-based programming tools also collect data at this level of granularity.

- *Version control systems* – some systems that focus on the state of the solution only, rather than event tracking, use version control systems to store histories of snapshots of the code being developed. This can be done with voluntary source code commits when students are actively using source code control on their projects, or by using automated instrumentation to transparently commit the state of the source code to a version control system.

- *Key logging* – at the finest level of granularity, some systems track and store events at the level of individual keystrokes. Effectively, the working environment has to be augmented with a keylogger. This technique appears to be more common in web-based problem solving environments at present, as well as newer IDE instrumentations. Here, there are also multiple levels of granularity — some contexts store progress, e.g. during a pause, leading to small bulk inserts, while others store each event individually.

There is no single "reference architecture" for how best to collect data as students work on exercises or assignments. Still, there are many common elements that recur across a number of research projects. To describe these common elements, Figure 4 illustrates a logical arrangement of key systems or subsystems that students interact with when developing solutions, and that instructors or researchers interact with when analyzing data. Few data collection tools currently exist that address all of the aspects of this logical architecture. However, Figure 4 provides a common foundation for describing how current tools both overlap and differ.

Students working on individual exercises or programming tasks primarily interact with tools for constructing and running program solutions, as well as for submitting and receiving assessment feedback on their work. These components are shown in Figure 4 (a)–(b). The interface students use for editing code as well as for executing or testing their work may be embodied in a stand-alone IDE, provided by a web interface, or even implemented as several distinct tools. Similarly, students may interact with a separate assessment system where they must explicitly submit their work, or the assessment capability may be built-in to their interactions with the development environment.

Underlying all of the "front end" components that students interact with, and where both event data and solution snapshots are often captured, data collection systems are centered around a data storage scheme, shown as (c) in Figure 4. Everything from simple flat files to centralized server-based database management systems have been used in various tools. When considering which tool may be most suitable for a given research task, thought should be given to the data storage implementation, as well as the specific data attributes that are recorded. In addition to data storage, some data collection tools also provide mechanisms for data export (d), which can simplify the workflow for data analysis. Some systems also provide for data summarization or visualization features (e), which can also be useful for both teachers and researchers.

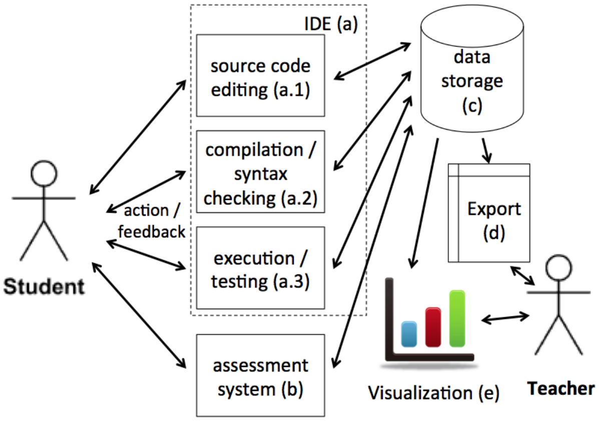

Figure 4: An abstract view of the logical services where data instrumentation and collection are typically performed—note that some data collection tools encompass multiple logical services.

Table 3 lists a number of currently active tools being used (or with significant potential to be used) for routine data collection in CS education research. The granularity of data collection at the various logical locations identified in Figure 4 are shown in the columns of Table 3. No specific mention of data storage (Figure 4 (c)) is indicated in the table, since all such data collection systems are built on top of a data storage infrastructure, most often a relational database.

Some of the tools listed in Table 3 are browser based whereas some let students use more commonly used programming environments. The use of browser based tools, in many cases, has focused on smaller scale homework problems (writing a single function or method) using a web interface. From a technical perspective, web based tools typically do not provide for separate user actions associated with saving. This implies that students cannot create intermediate versions where they could return later.

Finally, in addition to web-based practice problem systems, assessment tools, and IDE-based data collection, many other tools have arisen that show the many variations possible with the logical architecture shown in Figure 4. While Table 3 includes examples of those that are currently active, a number of others are summarized in Section 2.

4.2 How Data are Shared

Only a few data sets describing authentic solutions to programming problems are publicly available. One of the most well known is the Blackbox data set containing compilation events from hundreds of thousands of students [18]. There is no information on what tasks students are solving or any automated feedback in this data. Other publicly available data sets we were able to identify are from the code.org hour of code event [90] and Code Hunt game

Table 3: Examples of currently available (2015) programming data collection tools. For each tool we describe which programming languages we found are supported, what kind of UI/Client is provided for students, and at what granularity level data is collected on the client side (only the most fine grained level is named). All the systems listed here, except Blackbox, support automated assessment.

System	Languages	Client/Instrumentation
Blackbox [18]	Java	BlueJ/line edits
CloudCoder [85]	C/C++, Java, Python, Ruby	online ide/keystrokes
CodeWorkout [38]	Java, Python, Ruby	online ide/compilations
js-parsons [65]	Python, Pseudo code	online ide/drag and drop actions (D&D)
PCRS [118]	C, Python, RA, SQL	online ide/compilations
Problets [73]	C, C#, Java	online ide (Java applet)/compilations
TestMyCode [86]	Java	NetBeans/keystrokes
URI Online Judge [4]	C/C++, Java, Python	-
UUhistle [100]	Python, Java	online ide for visual program simulation/(D&D)
Web-CAT [39]	C++, Java, Python	-

environment [15]. Code.org data consist of approximately 500,000 learners solving two small programming tasks in a graphical environment where solutions are constructed by dragging and dropping code elements. AST presentations of all the execution points (submissions in the environment) together with test results are provided. Although the order of the execution points is known, there are no time-stamps attached to the events. Finally, names of the tasks are given but detailed exercise descriptions seen by the students are not available in this data set. The Code Hunt data is a preview set consisting of submissions generated by 258 users working on 24 small programming problems in Java and C#. Each submission contains source code, time stamp, exercise meta-data and an (anonymized) user-id. For each user, the data contains self-reported programming experience on scale 1="Beginner", 2="Intermediate", and 3="Advanced". The data sets are summarized in Table 4.

Each of these data sets is different. There are differences, for example, in programming environments, tasks, and how data is stored – unfiltered solution in Code Hunt, comments removed for anonymization in Blackbox, and AST only in code.org data. In addition, many of the differences are such that information provided in one data set is simply not available in another. For example, Code Hunt is the only data set providing information on students' previous programming experience.

In order to compare two data sets describing programming traces, the contexts from where data is collected as well as the granularity of the data are highly relevant. To better understand how these can differ, we shared a few data sets among the working group members. We found differences in programming languages, level of the studies, amount of data, tools being used to collect the data, granularity of the data, whether any information on student background and later performance is available, and at what level assignment meta-data is available. By assignment meta-data we mean assignment descriptions, test cases, feedback given for students, and information on pedagogy used on the course (e.g. what course material has been used). Characteristics of our data sets are given in Table 5. It is challenging to find nicely comparable data sets and even in data sets originating from our own environments, there are many details we simply don't know – e.g. as many teachers use CloudCoder we cannot know how these courses are structured.

4.3 Privacy and Ethical Concerns

As identified in the review of literature, very few studies made explicit mention of ethical and privacy issues. Nevertheless, since this work is focused on the collection and use of student programming process data, it is pertinent to acknowledge the myriad ethical and privacy issues surrounding this work.

4.3.1 Privacy

In this context, privacy relates to the ability to identify students from their programming process data, whether that is by the researcher collecting the data, or to the broader research community.

It is common practice to hide personal data in a data set. However, the identification of students can occur on multiple different levels. It is obvious that personal identifiers like name, student number, and email address have to be anonymised. However, students can also be identified by some set of their personal data (which is within a data set) like age, gender, organization, learning year, time stamp, IP-address used, etc. This is especially noteworthy when the size of the dataset is small. Additionally, students may be identifiable by the content they produce, which is laborious to check by hand. For example, source code files can include identifying information in comments, and the identifiers in the program code (student invented names of variables and data types) can contain information that may destroy the anonymity of the data set. Moreover, even the time between key presses can be used to distinguish between students [76]. All these aspects should be taken care of when data sets are prepared for publicity.

While these may seem like obvious concerns relating to the collection of programming process data, what is important to note is that only 4 out of the 76 studies discussed in the literature review explicitly mentioned privacy related issues. This is of concern for two primary reasons. First, if these issues were indeed addressed as part of the development of a tool or the collection of data, then it would be of significant benefit to the research community for them to be formally acknowledged. It not only adds to the rigour of the presentation of the research, but also provides important insight into key challenges in the automated collection and analysis of process data.

Secondly, if issues of privacy were not formally addressed

Table 4: Publicly available data sets with the URL for where they can be downloaded, number of users, number of tasks and number of data-points generated by the users working on all the tasks together in the published data sets.

Data	URL	#students	#tasks	#data points
Blackbox	http://www.bluej.org/blackbox.html	1M	NA	830M
Code Hunt	https://github.com/Microsoft/Code-Hunt	258	24	13K
code.org	http://stanford.edu/~cpiech/bio/pages/codedotorg/	500k	2	2.4M

Table 5: Characteristics of the data sets. Letters in code stand for D - Duke University, H - University of Helsinki, T - Toronto University, Y - York College. Assignment metadata column is coded so that a=assignment description, t=test cases, p=pedagogy, f=feedback.

Data	Lang	level	students	#tasks	tool*	demo- graphics	course grade	assignment metadata	granularity
Blackbox	Java	N/A	1M	*	ide	yes	no	-	line edits
PCRS	Python	CS-1	3300	60	online ide	no	yes	atpf	submissions
Code Hunt	Java/C#	N/A	258	24	online ide	no	no	at–	submissions
CloudCoder (D13)	Python	CS-1	233	62	online ide	no	yes	at-f	keystrokes
CloudCoder (D14)	Python	CS-1	194	55	online ide	no	yes	at-f	keystrokes
CloudCoder (Y13)	C	CS-1	133	62	online ide	no	yes	at-f	keystrokes
CloudCoder (Y14)	C	CS-1	86	53	online ide	no	yes	at-f	keystrokes
Marmoset (M06)	Java	CS-2	96	6	online ide	no	no		file saves
code.org	Blockly	CS-0	500K	2	N/A	no		N/A	submissions
TMC (H14)	Java	CS-1	150	100+	ide	yes	yes	atp-	keystrokes

as part of the research study, then this leads to many other concerns. The ethical nature of the research can be brought into question if this is the case, which we discuss further below. However, at a practical level, the ability to extend on the work being undertaken may become compromised. An identifiable dataset cannot be feasibly shared or used by different researchers. This may lead to an inability to appropriately share details of the approach in order for the research to be replicated or reproduced. The work of Brown et al. [18] regarding Blackbox is a good example of how research in this area can identify ethical challenges and address them in ways to potentially facilitate the growth of the field. However, the lack of acknowledgment of the privacy issues in most research may be one of the greatest reasons to date for a lack of reproduction research in this field.

4.3.2 Ethics

Undeniably related to the privacy issues raised are issues relating to research ethics. Ethical issues can relate not only to the collection of data and transparency to participants, but also to the use of this data, in particular in how it can influence the learning relationship with students. When collecting data related to students it is important to be aware of the ethical guidelines and regulations. They exist to protect students' privacy and security. Disclosing a student's learning history can influence his/her career and even social life. Additionally, objective grading must not be compromised.

Appropriate scientific practice is typically controlled by the Ethics Committee of the university where the research originally takes place. The researcher needs to get permission to gather and use such data in advance. The bureaucracy related to this varies from university to university. Adcock et al. [5] describe drawing the line regarding the need of ethical permission in their university as follows:

> "The most important non-technical issue we faced was a consequence of our mere intent to use logged data in research to be reported to the CS education community. This meant that an institutionally-approved human subjects protocol was required before data collection could even begin. If we had decided to use the error logs only for local instructional purposes (e.g., to identify for special attention students having the most trouble on their programming assignments), then no such requirement would have been imposed."

This example also highlights a considerable problem in the ability to build upon the work conducted by other researchers. Institutional policies for protection of data and individuals exist and will undoubtedly continue to exist. These can often be accommodated by eliminating data that can be considered identifiable in any way. However, future research in the area of collection of programming process data should seek to consider how data could be collected to contain enough granularity to be meaningful, yet be in a completely unidentifiable form (not only to protect the student but possibly also the learning institution). This could facilitate data sharing and the building of immense data sets for analysis. We consider this to be a grand challenge for this research area.

The other facet of ethical consideration is for what purpose the data is used in relation to the students learning. Purposes such as building understanding of student learning behaviours in general does not pose considerable concern. Using the data for providing intelligent tutoring or guidance may begin to raise some ethical concerns, as this can infer that a system may be pre-preemptively guiding a student's behaviour. How much should an automated system interact with the students' natural learning process? While there are undoubtedly benefits to providing such supportive mechanisms, it may be difficult to balance the use of these technologies with the need to use data for the purpose of student assessment.

5. CASE STUDIES: RE-ANALYSIS, REPLICATION AND REPRODUCTION

In order to better understand the challenges that might lead to the lack of studies that attempt to replicate or reproduce previously published work, our working group performed three case studies related to re-analysis, replication and reproduction of existing work. Our intention here is not to carry out three separate studies of the quality expected from a conference publication. Rather, we explore and report on the challenges and difficulties encountered in replicating previously published work, and we illustrate the impact of confounding factors, such as differences in development environment or language, on results.

5.1 Re-analysis: A different research team verifies results

Spacco et al. [101] performed a post-hoc analysis of data collected by CloudCoder over five semesters across three different institutions. Their results show that, as students do more exercises, their scores on the exercises tend to decrease, although the likelihood of submitting correctly compiling code increases. Presumably, this is because the exercises tend to be more difficult at the end of the term. However, the mastery of syntax is, likely, primarily effected by practice, not the difficulty of the code to be written.

Working group members who were not authors of the original paper re-analyzed Spacco et al.'s results. This included two regression analyses, which were performed using two software packages to validate the obtained results (R and Microsoft Excel).

Re-analysis was performed without any personal communication with the original authors. Under these conditions, it was possible to recreate both the evolution of students' score from exercises as well as the evolution of correctly compiling code for datasets (Figure 5 describes the evolution of students' best scores over time). A bootstrapping analysis was run on the results, which were shown not to be significantly different from those obtained by Spacco et al. [101]. Asking the original authors a series of clarifying questions improved the results only marginally without any significant difference. The authors then tried to perform the results with a third data set using the previous assumptions, but they failed. This indicates that re-analysis is a manual process and that each assumption must be checked for each used dataset.

During the re-analysis process, it was learned that while it is important to know what specific tools were used for the data analysis (R, SPSS, PyLab, etc.), actually using those tools is not necessary as the provided implementations in different tools are very similar. It was also discovered that the presence of metadata that describes different rows and columns of each dataset is an absolute necessity, as different datasets may need different types of pre-processing.

Another challenge that the working group members faced was trying to calculate statistical estimators. It was learned that it is important to have good definitions of the sets used in the statistical calculations. An example of this would be the calculation of the mean of the best score of each exercise which can be calculated both by including or excluding the students that attempted exercise but never compiled it successfully. Finally, we noticed that it is important to specify the parameters in detail. For example, a successful compilation event might be interpreted differently as it is a context-dependent term which might vary depending on the tool used in the study.

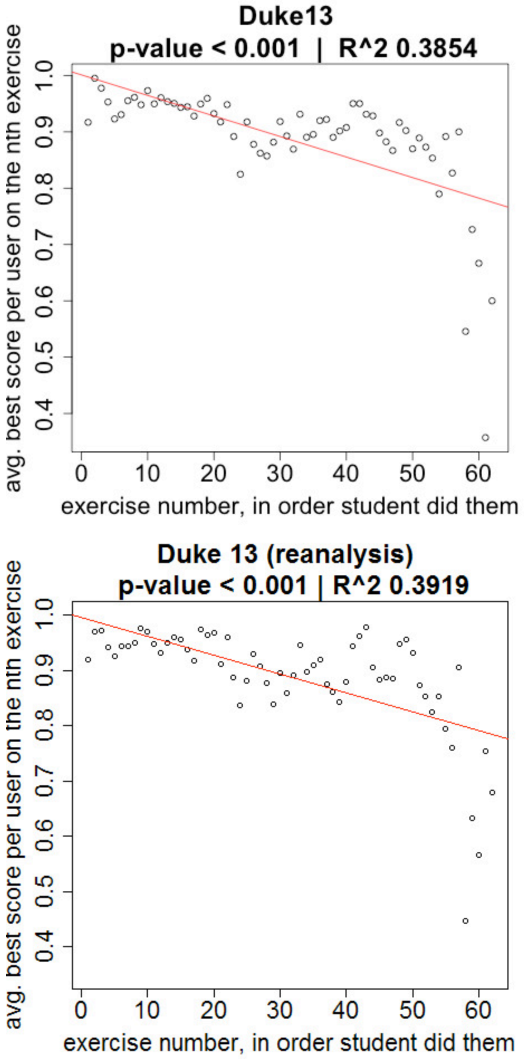

Figure 5: Evolution of the best scores achieved by each student along time. Top: Original results. Bottom: Re-analysis

5.2 Replication: Applying a previously published analysis method to a different data set

We then tried to verify the analysis of Spacco et al. [101] using a different data set than the one used for the original paper. For this replication, we used data collected by PCRS [118] over two semesters at the University of Toronto Mississauga. PCRS is a browser-based system that provides a number of different features from CloudCoder, but does collect time-stamped submissions and the results of test cases, which is of the same format as the data used in the original study.

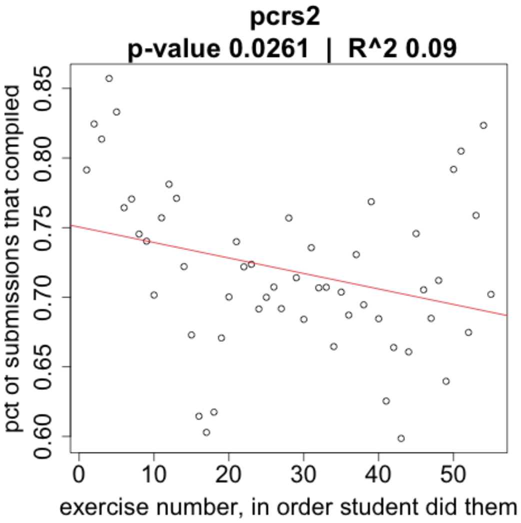

Figure 6: The percentage of compilable submissions as a user works on more programming exercise.

The original CloudCoder data suggested that the likelihood that a student's Python submission compiles correctly tends to increase with each new exercise a student attempts. However, we can see in Figure 6 that this finding was not present in the PCRS dataset, as the percentage of successful compilations decreases over time. Upon further analysis of the factors that could explain this phenomenon, we observed that the early exercises in PCRS are smaller than those in CloudCoder.

In addition, we observed that while both CloudCoder and PCRS Python exercises are functions that require students to fill in the body of the function, and both systems also allow the instructor to provide a skeleton or template, the template code from the PCRS data set always compiles correctly (i.e. it contains "return 0" or "return null"), while the CloudCoder templates have an empty method body that does *not* compile. It is possible that even this simple decision has lead to the previously observed results.

It is important to highlight that a seemingly small detail that was omitted in the original paper by Spacco et al., i.e., whether the instructor-provided code templates compile, or even whether templates are provided at all, turned out to be of great importance, not only to replicate the study with new data, but also to fully understand the results of the original work. Before attempting to replicate the study with a separate dataset, we were unaware that this detail may have been important.

5.3 Reproduction: Compilation Behaviour

Jadud conducted a series of studies of novice compilation behavior in a Java-based CS1 course at a research-intensive European university [59–61,94]. The work instrumented the BlueJ IDE to report every compilation performed by a student, and used this data to describe how novices interact with the compiler and the errors they encounter. An error quotient (EQ) is proposed as a measure to predict student performance. A reproduction of EQ has been published elsewhere [87], but for the purpose of our case study, we have found it instructive to attempt to reproduce contextual data reported in the 2006 study [61] using a different data set and researcher.

5.3.1 Comparison of Data Sources

For the reproduction, we use a data set generated in a CS1 course at a large, North American R1 university. The data set contains a set of small, weekly coding problems presented to students. The original paper analyzes 42,000 compilations spread across 2,100 sessions [61]. The new data set contains 48,037 submissions spread across 2,994 sessions. 130 students were observed in the original study, and the new data set includes submissions from 476 students.

This data set is different from the original set in two main ways. First, the original data set was based on Java assignments, whereas the new data set features Python exercises. Unlike Java, which is compiled, Python is an interpreted language with errors reported at runtime. To produce a comparison, we switch from the idea of evaluating "compilation" to evaluation of a "submission". Each submission causes the code to be executed and for feedback to be returned, including errors and, if the code is syntactically correct, the results of tests. Similar to the original setup, only a single error is returned when the code is executed.

Second, in the original data set, students were adding small pieces of functionality to multiple locations in a starter code template. The starter code is fairly large – spanning multiple files. In contrast, in the new set, students are submitting small, independent exercises. Since the exercises are independent, failure to "compile" one exercise does not keep a student from making progress on a different exercise. It should be easier for a student to stop and move to another location in the code (for us, a different exercise), as Jadud observed in the original study.

5.3.2 Defining Sessions

The Jadud work relies on the idea of a work session. Each session is a sequence of at least seven compilations of different (edited) code where the student has not exited the IDE. Sessions with less than seven compilation events are discarded. The paper that first reports EQ results describes seven compilations as "a tested and reasonable cut-off for defining the length of a session", but the specific process for selecting the cut-off are not discussed [61]. Jadud's thesis discusses the issue in the context of data filtering, with the aim of generating a "representative sample" [59].

We must provide a different definition for a work session that reflects what we assume was the original intent: a contiguous period of time when a student is focused on working on the assignment. We define a work session to be *a sequence of at least seven distinct (edited) submissions, for any number of exercises, where no neighboring submissions in the sequence are separated by more than 20 minutes.* The value of 20 minutes was obtained by exploration. There is a gap of roughly between 15 minutes and one hour where relatively few new sessions are created.

5.3.3 Comparing Contextual Results

The introduction to the EQ paper presents three figures as background that are similar to those presented from a different context (from compilations observed in a lab setting) [60]. The figures reveal differences that result from the contexts from which the data sets were drawn.

As expected, the two languages report errors in very dif-

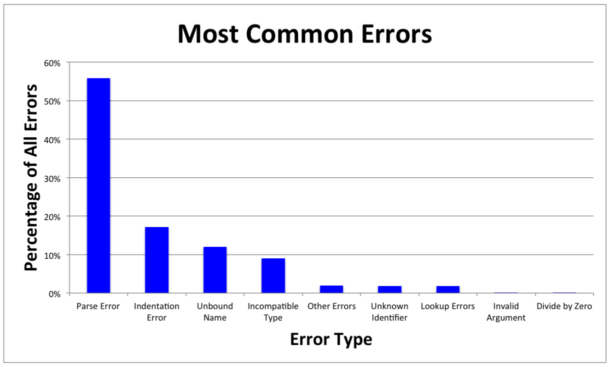

Figure 7: The most common errors encountered by students using Python. (Compare to Figure 1 in [61].)

ferent ways. Figure 7, which reproduces Figure 1 in the EQ paper, describes the errors encountered by students using Python. The number of Parse Errors encountered in Python is very high. While Java reports specific issues, like missing semicolons and mismatched brackets (that are reported in separate columns in Jadud's figure), Python uses a single error – SyntaxError (a Parse Error) – for any issue that causes parsing to fail. The table also contains several inclusions and omissions. Type-related issues that Java might report during compile time may not generate any error at all in Python, while Lookup and Zero Division errors reported by Python are runtime errors in Java.

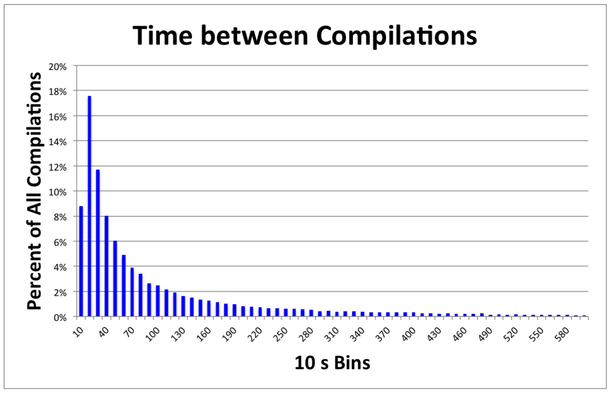

Figure 8: Time between pairs of compilations. (Compare to Figure 2 in [61].)

Figure 8 reproduces the second figure in the EQ paper. Both figures illustrate how commonly students rapidly recompile, with roughly 38% of compilations in the Python data set (and 35% in Java) occurring within the first 30 seconds and a rapid decline thereafter. However, the peak submission time in the Python set occurs at 20 seconds, compared to 10 seconds in the Java set.

Figure 3 in the EQ paper makes the point that students recompile quickly when they have a syntax error to correct. However, as illustrated in Figure 9, the change from "compilations" to "submissions" is significant. Students receive feedback not only about the syntactic correctness of their submission but also about its functional correctness. Hence, students in the Python data set resubmit code quickly regardless of whether it "compiles" or not. This suggests that students resubmit quickly regardless of the type of error (compilation or logical) they are solving.

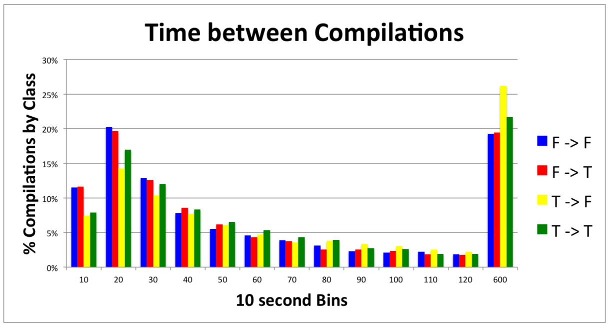

Figure 9: Time between pairs of compilations by compilation state (T compiles, F fails to compile). (Compare to Figure 3 in [61].)

Jadud presents an algorithm for computing the Error Quotient (EQ), a heuristic that "characterizes how much or how little a student struggles with syntax errors while programming" [61]. As might be expected from the differences in context illustrated in the previous three figures, the process for computing EQ does not directly transfer to a new context. We performed the filtering process described in the original paper to eliminate students with little data and calculated EQ for each session using the weights in the algorithm provided. A linear model estimation found that our EQ scores were correlated neither with assignment nor exam marks at the $p < 0.05$ significant level. A major challenge in this study is thus the sometimes unspecified differences in context, which might lead to wrong assumptions or measures being used to understand or predict student behavior.

6. DISCUSSION AND CONCLUSIONS

In this working group report, we have outlined the current state of the field of educational data mining and learning analytics of how students solve programming problems. During the last ten years, there has been a substantial increase in work in the field, which is observable from the quantity of relevant articles. Despite this, our survey suggests there is a lack of multi-institutional work such as [81, 108] in educational data mining and learning analytics in programming. In addition, whilst many tools and techniques used for collecting data exist, results in our field are rarely backed up with raw data made publicly available. Although researchers have their own datasets, studies that seek to verify and extend previous results are still scarce. To better understand why this is the case, we conducted three cases studies on reanalysis, replication and reproduction of some well known studies in our field.

During re-analysis of previous results, where new researchers analyzed existing data with existing methodology, two out of three data sets produced comparable results to those in an original study, while issues related to lacking documentation caused issues with the re-analysis of the third data set. This was followed by a replication, where previously published results were sought after by a new researcher with a new but similar dataset. Whilst the replication failed to reach the same conclusions as the original article, additional factors that could explain the original results were identified, thus providing an example of why replication studies are important for the field. The third case study considered the reproduction of a well-known metric using a different data set, different researcher, and altered methodology – that is, the study sought to identify whether the metric is observable even if much of the analysis changes. The reproduction process was challenging, and, in the end, we failed to reproduce the previously observed results. This does not mean that the phenomenon does not exist, but that the details of the context contribute to the outcomes.

Our field is not the only one where reproducibility is a challenge. In a recent study, reproduction of many well known psychological studies failed when attempted in a new context [28]. The inability to repeat experiments and analyses introduces a number of challenges even in the core data mining community where Blockeet and Vanshoren [16], for example, have proposed the creation of an "experiment database" to facilitate replication. In particular, they state that: "*...it should be clear how the experiments can be reproduced. This involves providing a complete description of both the experimental setup (which algorithms to run with which parameters on which datasets, including how these settings were chosen) and the experimental procedure (how the algorithms are run and evaluated). Since space is limited in paper publications, an online log seems the most viable option.*" [16]. An "experiment database" and/or an "online log" have a number of attractive features. These include the evaluation of new experimental techniques, discussing experiments that did not make it to publication, making it easier to meet the page limits of conferences, researchers being able to check existing results and work more quickly, and being able to identify possible errors in their designs more rapidly due to access to additional comparison points.

Drummond [35] responds to Blockeel and Vanschoren by highlighting the practical difficulty of an experiment database: "*[s]urely we wouldn't expect reviewers to very carefully study the scripts etc needed to produce the results. Yet, simply checking that they can reproduce the tables and graphs of the paper would seem to do little to validate the work.*" [35]. These perspectives raise important philosophical, technical, and policy questions. Requiring authors to include an "online log" of their complete experiment would likely simplify replication. However, an "online log" raises a number of questions of its own, for example, how much time and effort should go into producing the online log? If a paper is rejected from one conference with one log format, how much effort is expected for authors to modify the log for a new conference? Are the reviewers expected to understand and review the logs as well? Finally, as our field falls in between human sciences and data mining, the challenges of reporting contextual details are multi-faceted. On one hand, only reporting parameters and used algorithms is not sufficient as details are needed also for data preparation and extraction, while on the other hand, as we do not know the confounding factors, providing all the details on the context is not feasible.

There is a fundamental tension between innovation and standardization in academia, especially in new, rapidly evolving fields such as educational data mining. We want to provide the maximum flexibility for researchers to create innovative algorithms and approaches to analyze and interpret data, but we also want a standardized language for researchers to express their innovations such that other research groups could replicate their findings. Too much unchecked innovation may hinder replication, but premature standardization may stifle creativity. Perhaps at this moment, in 2015, a typical conference paper, with its limited number of pages and lengthy obligatory sections (references, related work, introduction) leaves too little room to provide sufficient detail for easy replication. However, does this mean that replication studies will still be rare in 2025?

7. FUTURE DIRECTIONS

The way the computer science education field will be perceived ten years from now depends heavily on the actions of our field today. To show a glimpse of our vision of the future state of the field, we outline five Grand Challenges – or goals – for researchers and practitioners.

As observed in the case study on repetition, where previously published analysis methods were applied to a different data sets, context-specificity may lead to unfounded or erroneous conclusions. This is because the educational landscape varies drastically from country to country, and even within the same institution and course, as the pedagogical approaches and used programming languages change. Ignoring context changes such as these leads to conclusions from one study being wrongfully applied in drastically different contexts. As the majority of results, so far, come from studies on a single institution and course, the first Grand Challenge is to *have researchers and practitioners commit to building and maintaining a multi-language, multi-institution, multi-nation learning process data and experiment result database* that contains student metadata, logs from systems that the students use, and other details, which could then be used as a reflection point for novel results from a specific context. At the same time, as pointed out in the literature review, privacy issues need more attention. Here, with the ever-increasing detail of data, maintaining participant privacy becomes increasingly challenging.

Our analysis also revealed that the majority of studies did not consider or report upon confounding factors that may contribute to the observed outcomes. Whilst assumptions are made based on the data at the disposal of the researchers, understanding results from previous work is needed in order for the field to move forward. Thus the second Grand Challenge is to *systematically analyze and verify previous studies using data from multiple contexts to tease out tacit factors that contribute to previously observed outcomes.* Here, the R.A.P. Taxonomy introduced in Section 3.2. provides a guideline for the types of studies that are needed.

Whilst the first two Grand Challenges are related to exploring existing and future data sets to build a solid base of knowledge, we expect that the field will also evolve in how research is conducted. Many of the studies found in the literature review were post-hoc studies of data from a single context, and very few papers reported on experimental settings or pilots where some conditions were varied. Whilst post-hoc studies are a part of an exploratory field, we need to refrain from drawing strong conclusions from such studies, especially if the data comes from a single context or course. Instead, scientific experimental methods should be adopted to a larger degree. Towards this, the third Grand Challenge is to *use pilots and experiments, with control and treatment groups, to evaluate and explain the results.*

The post-hoc nature of the studies as discussed above also influences their applicability. For example, a number of subcategories in Table 2, such as *Drop-out risk and performance*, which identifies students who are at risk of dropping out, have been studied to an extent, but have not been employed to create actual interventions where some of the proposed methods would be put into practice. We believe that theoretical results are important, but that in this field their real value comes from application. The fourth Grand Challenge is to *adopt results and practices into classroom use to continuously monitor and improve offered education.* Through in-situ observations and emerging practices, practitioners will help others to adapt the presented practices to their contexts, which will invigorate the field, and open up completely new research streams.

The final challenge relates to generalization. Our literature review found that very few studies underpinned their results or analysis methods on a specific theory or a model of pedagogy or educational practices. This implies, together with the drawbacks identified above, that results and practices will be more difficult if not impossible to generalize. Thus, when initial results have been received from adopting practices into classrooms, the next step, and fifth Grand Challenge, is to *generalize the results to other contexts, if possible, and help practitioners apply them in their respective fields.*

Acknowledgements

The authors would like to thank Karen Petrie for her contributions to the working group discussions.

8. REFERENCES

[1] CodingBat. http://codingbat.com/help.html\#teacher. Accessed: 2015-07-08.

[2] edX XServer. https://github.com/edx/xserver. Accessed: 2015-07-07.

[3] TuringsCraft CodeLab. http://www.turingscraft.com/. Accessed: 2015-07-07.

[4] URI online judge. https://www.urionlinejudge.com.br/academic/login. Accessed: 2015-07-08.

[5] B. Adcock, P. Bucci, W. D. Heym, J. E. Hollingsworth, T. Long, and B. W. Weide. Which Pointer Errors Do Students Make? In *Proceedings of the 38th SIGCSE Technical Symposium on Computer Science Education*, SIGCSE '07, pages 9–13, New York, NY, USA, 2007. ACM.

[6] M. Ahmadzadeh, D. Elliman, and C. Higgins. An Analysis of Patterns of Debugging Among Novice Computer Science Students. *SIGCSE Bull.*, 37(3):84–88, June 2005.

[7] A. Alammary, A. Carbone, and J. Sheard. Implementation of a Smart Lab for Teachers of Novice Programmers. In *Proceedings of the Fourteenth Australasian Computing Education Conference - Volume 123*, ACE '12, pages 121–130, Darlinghurst, Australia, Australia, 2012. Australian Computer Society, Inc.

[8] A. Allevato, S. H. Edwards, and M. A. Pérez Qui nones. Dereferee: Exploring Pointer Mismanagement in Student Code. In *Proceedings of the 40th ACM Technical Symposium on Computer Science Education*, SIGCSE '09, pages 173–177, New York, NY, USA, 2009. ACM.

[9] A. Allevato, M. Thornton, S. Edwards, and M. Perez-Quinones. Mining data from an automated grading and testing system by adding rich reporting capabilities. In *Educational Data Mining*, pages 167–176, 2008.

[10] A. Altadmri and N. C. C. Brown. 37 Million Compilations: Investigating Novice Programming Mistakes in Large-Scale Student Data. In *Proceedings of the 46th ACM Technical Symposium on Computer Science Education*, SIGCSE '15, pages 522–527, New York, NY, USA, 2015. ACM.

[11] P. Antonucci, C. Estler, D. Nikolić, M. Piccioni, and B. Meyer. An Incremental Hint System For Automated Programming Assignments. In *Proceedings of the 2015 ACM Conference on Innovation and Technology in Computer Science Education*, ITiCSE '15, pages 320–325, New York, NY, USA, 2015. ACM.

[12] P. Bancroft and P. Roe. Program Annotations: Feedback for Students Learning to Program. In *Proceedings of the 8th Australasian Conference on Computing Education - Volume 52*, ACE '06, pages 19–23, Darlinghurst, Australia, Australia, 2006. Australian Computer Society, Inc.

[13] M. Berges and P. Hubwieser. Evaluation of Source Code with Item Response Theory. In *Proceedings of the 2015 ACM Conference on Innovation and Technology in Computer Science Education*, ITiCSE '15, pages 51–56, New York, NY, USA, 2015. ACM.

[14] J. L. Bez, N. Tonin, and P. R. Rodegheri. URI Online Judge Academic: A tool for algorithms and programming classes. In *Computer Science &amp; Education (ICCSE), 2014 9th International Conference on*, pages 149–152. IEEE, Aug. 2014.

[15] J. Bishop, R. N. Horspool, T. Xie, N. Tillmann, and J. de Halleux. Code Hunt: Experience with Coding Contests at Scale. In *Proceedings of the 37th International Conference on Software Engineering - Volume 2*, ICSE '15, pages 398–407, Piscataway, NJ, USA, 2015. IEEE Press.

[16] H. Blockeel and J. Vanschoren. Experiment Databases: Towards an Improved Experimental Methodology in Machine Learning. In J. Kok, J. Koronacki, R. Lopez de Mantaras, S. Matwin, D. Mladenič, and A. Skowron, editors, *Knowledge Discovery in Databases: PKDD 2007*, volume 4702 of *Lecture Notes in Computer Science*, pages 6–17. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007.

[17] N. C. C. Brown and A. Altadmri. Investigating Novice Programming Mistakes: Educator Beliefs vs. Student Data. In *Proceedings of the Tenth Annual Conference on International Computing Education Research*, ICER '14, pages 43–50, New York, NY, USA, 2014. ACM.

[18] N. C. C. Brown, M. Kölling, D. McCall, and I. Utting. Blackbox: A Large Scale Repository of Novice Programmers' Activity. In *Proceedings of the 45th ACM Technical Symposium on Computer Science Education*, SIGCSE '14, pages 223–228, New York, NY, USA, 2014. ACM.

[19] P. Brusilovsky, S. Edwards, A. Kumar, L. Malmi, L. Benotti, D. Buck, P. Ihantola, R. Prince, T. Sirkiä, S. Sosnovsky, J. Urquiza, A. Vihavainen, and M. Wollowski. Increasing Adoption of Smart Learning Content for Computer Science Education. In *Proceedings of the Working Group Reports of the 2014 on Innovation & Technology in Computer Science Education Conference*, ITiCSE-WGR '14, pages 31–57, New York, NY, USA, 2014. ACM.

[20] K. Buffardi and S. H. Edwards. Exploring Influences on Student Adherence to Test-driven Development. In *Proceedings of the 17th ACM Annual Conference on Innovation and Technology in Computer Science Education*, ITiCSE '12, pages 105–110, New York, NY, USA, 2012. ACM.

[21] K. Buffardi and S. H. Edwards. Effective and Ineffective Software Testing Behaviors by Novice Programmers. In *Proceedings of the Ninth Annual International ACM Conference on International Computing Education Research*, ICER '13, pages 83–90, New York, NY, USA, 2013. ACM.

[22] K. Buffardi and S. H. Edwards. Impacts of Adaptive Feedback on Teaching Test-driven Development. In *Proceeding of the 44th ACM Technical Symposium on Computer Science Education*, SIGCSE '13, pages 293–298, New York, NY, USA, 2013. ACM.

[23] K. Buffardi and S. H. Edwards. A Formative Study of Influences on Student Testing Behaviors. In *Proceedings of the 45th ACM Technical Symposium on Computer Science Education*, SIGCSE '14, pages 597–602, New York, NY, USA, 2014. ACM.

[24] K. Buffardi and S. H. Edwards. Responses to Adaptive Feedback for Software Testing. In *Proceedings of the 2014 Conference on Innovation and Technology in Computer Science Education*, ITiCSE '14, pages 165–170, New York, NY, USA, 2014. ACM.

[25] J. Carter, P. Dewan, and M. Pichiliani. Towards Incremental Separation of Surmountable and Insurmountable Programming Difficulties. In *Proceedings of the 46th ACM Technical Symposium on Computer Science Education*, SIGCSE '15, pages 241–246, New York, NY, USA, 2015. ACM.

[26] B. Cheang, A. Kurnia, A. Lim, and W.-C. Oon. On automated grading of programming assignments in an academic institution. *Computers & Education*, 41:121–131, 2003.

[27] Y. Cherenkova, D. Zingaro, and A. Petersen. Identifying Challenging CS1 Concepts in a Large Problem Dataset. In *Proceedings of the 45th ACM Technical Symposium on Computer Science Education*, SIGCSE '14, pages 695–700, New York, NY, USA, 2014. ACM.

[28] O. S. Collaboration and Others. Estimating the reproducibility of psychological science. *Science*, 349(6251), 2015.

[29] P. Denny, A. L. Reilly, and D. Carpenter. Enhancing Syntax Error Messages Appears Ineffectual. In *Proceedings of the 2014 Conference on Innovation & Technology in Computer Science Education*, ITiCSE '14, pages 273–278, New York, NY, USA, 2014. ACM.

[30] P. Denny, A. L. Reilly, and E. Tempero. All Syntax Errors Are Not Equal. In *Proceedings of the 17th ACM Annual Conference on Innovation and Technology in Computer Science Education*, ITiCSE '12, pages 75–80, New York, NY, USA, 2012. ACM.

[31] P. Denny, A. L. Reilly, E. Tempero, and J. Hendrickx. CodeWrite: Supporting Student-driven Practice of Java. In *Proceedings of the 42nd ACM Technical Symposium on Computer Science Education*, SIGCSE '11, pages 471–476, New York, NY, USA, 2011. ACM.

[32] P. Denny, A. L. Reilly, E. Tempero, and J. Hendrickx. Understanding the Syntax Barrier for Novices. In *Proceedings of the 16th Annual Joint Conference on Innovation and Technology in Computer Science Education*, ITiCSE '11, pages 208–212, New York, NY, USA, 2011. ACM.

[33] A. K. Dominguez, K. Yacef, and J. R. Curran. Data Mining for Individualised Hints in e-Learning. In *Proceedings of the International Conference on Educational Data Mining*, pages 91–100, 2010.

[34] S. H. Downs and N. Black. The feasibility of creating a checklist for the assessment of the methodological quality both of randomised and non-randomised studies of health care interventions. *Journal of Epidemiology and Community Health*, 52(6):377–384, June 1998.

[35] C. Drummond. Replicability is not Reproducibility: Nor is it Good Science. In *Proceedings of the Evaluation Methods for Machine Learning Workshop at the 26th International Conference for Machine Learning*, 2009.

[36] T. Dy and Ma. A Detector for Non-literal Java Errors. In *Proceedings of the 10th Koli Calling International Conference on Computing Education Research*, Koli Calling '10, pages 118–122, New York, NY, USA, 2010. ACM.

[37] G. Dyke. Which Aspects of Novice Programmers' Usage of an IDE Predict Learning Outcomes. In *Proceedings of the 42Nd ACM Technical Symposium on Computer Science Education*, SIGCSE '11, pages 505–510, New York, NY, USA, 2011. ACM.

[38] S. Edwards and K. Buffardi. CodeWorkout. http://web-cat.org/group/codeworkout. Accessed: 2015-07-07.

[39] S. H. Edwards and M. A. Perez-Quinones. Web-CAT: Automatically Grading Programming Assignments. *SIGCSE Bull.*, 40(3):328, June 2008.

[40] S. H. Edwards and Z. Shams. Do Student Programmers All Tend to Write the Same Software Tests? In *Proceedings of the 2014 Conference on Innovation & Technology in Computer Science Education*, ITiCSE '14, pages 171–176, New York, NY, USA, 2014. ACM.

[41] S. H. Edwards, Z. Shams, M. Cogswell, and R. C. Senkbeil. Running Students' Software Tests Against Each Others' Code: New Life for an Old "Gimmick". In *Proceedings of the 43rd ACM Technical Symposium on Computer Science Education*, SIGCSE '12, pages 221–226, New York, NY, USA, 2012. ACM.

[42] S. H. Edwards, Z. Shams, and C. Estep. Adaptively Identifying Non-terminating Code when Testing Student Programs. In *Proceedings of the 45th ACM Technical Symposium on Computer Science Education*, SIGCSE '14, pages 15–20, New York, NY, USA, 2014. ACM.

[43] S. H. Edwards, J. Snyder, M. A. Pérez Qui nones, A. Allevato, D. Kim, and B. Tretola. Comparing Effective and Ineffective Behaviors of Student Programmers. In *Proceedings of the Fifth International Workshop on Computing Education Research Workshop*, ICER '09, pages 3–14, New York, NY, USA, 2009. ACM.

[44] N. Falkner, R. Vivian, D. Piper, and K. Falkner. Increasing the Effectiveness of Automated Assessment by Increasing Marking Granularity and Feedback Units. In *Proceedings of the 45th ACM Technical Symposium on Computer Science Education*, SIGCSE '14, pages 9–14, New York, NY, USA, 2014. ACM.

[45] N. J. G. Falkner and K. E. Falkner. A Fast Measure for Identifying At-risk Students in Computer Science. In *Proceedings of the Ninth Annual International Conference on International Computing Education Research*, ICER '12, pages 55–62, New York, NY, USA, 2012. ACM.

[46] J. B. Fenwick, C. Norris, F. E. Barry, J. Rountree, C. J. Spicer, and S. D. Cheek. Another Look at the Behaviors of Novice Programmers. *SIGCSE Bull.*, 41(1):296–300, Mar. 2009.

[47] O. S. Gómez, N. Juristo, and S. Vegas. Replication, reproduction and re-analysis: Three ways for verifying experimental findings. In *Proceedings of the 1st international workshop on replication in empirical software engineering research (RESER 2010), Cape Town, South Africa*, 2010.

[48] O. Gotel, C. Scharff, and A. Wildenberg. Extending and Contributing to an Open Source Web-based System for the Assessment of Programming Problems. In *Proceedings of the 5th International Symposium on Principles and Practice of Programming in Java*, PPPJ '07, pages 3–12, New York, NY, USA, 2007. ACM.

[49] L. Grandell, M. Peltomäki, and T. Salakoski. High school programmingâĂŤa beyond-syntax analysis of novice programmers' difficulties. In *Proceedings of the Koli Calling 2005 Conference on Computer Science Education*, pages 17–24, 2005.

[50] L. Haaranen, P. Ihantola, L. Hakulinen, and A. Korhonen. How (Not) to Introduce Badges to Online Exercises. In *Proceedings of the 45th ACM Technical Symposium on Computer Science Education*, SIGCSE '14, pages 33–38, New York, NY, USA, 2014. ACM.

[51] W. Hartmann, J. Nievergelt, and R. Reichert. Kara, finite state machines, and the case for programming as part of

general education. In *Human-Centric Computing Languages and Environments, 2001. Proceedings IEEE Symposia on*, pages 135–141, 2001.

[52] J. Helminen, P. Ihantola, and V. Karavirta. Recording and Analyzing In-browser Programming Sessions. In *Proceedings of the 13th Koli Calling International Conference on Computing Education Research*, Koli Calling '13, pages 13–22, New York, NY, USA, 2013. ACM.

[53] J. Helminen, P. Ihantola, V. Karavirta, and L. Malmi. How Do Students Solve Parsons Programming Problems?: An Analysis of Interaction Traces. In *Proceedings of the Ninth Annual International Conference on International Computing Education Research*, ICER '12, pages 119–126, New York, NY, USA, 2012. ACM.

[54] C. Higgins, T. Hegazy, P. Symeonidis, and A. Tsintsifas. The CourseMarker CBA System: Improvements over Ceilidh. *Education and Information Technologies*, 8(3):287–304, Sept. 2003.

[55] C. A. Higgins, G. Gray, P. Symeonidis, and A. Tsintsifas. Automated Assessment and Experiences of Teaching Programming. *J. Educ. Resour. Comput.*, 5(3), Sept. 2005.

[56] R. Hosseini, A. Vihavainen, and P. Brusilovsky. Exploring problem solving paths in a Java programming course. In *Psychology of Programming Interest Group Annual Conference 2014*, page 65, 2014.

[57] D. Hovemeyer and J. Spacco. CloudCoder: A Web-based Programming Exercise System. *J. Comput. Sci. Coll.*, 28(3):30, Jan. 2013.

[58] M. C. Hughes, M. C. Jadud, and M. Rodrigo. String formatting considered harmful for novice programmers. *Computer Science Education*, 20(3):201–228, Sept. 2010.

[59] M. Jadud. *An Exploration of Novice Compilation Behaviour in BlueJ*. PhD thesis, 2006.

[60] M. C. Jadud. A First Look at Novice Compilation Behaviour Using BlueJ. *Computer Science Education*, 15(1):25–40, Mar. 2005.

[61] M. C. Jadud. Methods and Tools for Exploring Novice Compilation Behaviour. In *Proceedings of the Second International Workshop on Computing Education Research*, ICER '06, pages 73–84, New York, NY, USA, 2006. ACM.

[62] P. M. Johnson. Requirement and Design Trade-offs in Hackystat: An In-Process Software Engineering Measurement and Analysis System. In *Empirical Software Engineering and Measurement, 2007. ESEM 2007. First International Symposium on*, pages 81–90. IEEE, 2007.

[63] P. M. Johnson, H. Kou, J. M. Agustin, Q. Zhang, A. Kagawa, and T. Yamashita. Practical automated process and product metric collection and analysis in a classroom setting: lessons learned from Hackystat-UH. In *Proceedings of the International Symposium on Empirical Software Engineering*, pages 136–144. IEEE, Aug. 2004.

[64] A. Karahasanović and R. C. Thomas. Difficulties Experienced by Students in Maintaining Object-oriented Systems: An Empirical Study. In *Proceedings of the Ninth Australasian Conference on Computing Education - Volume 66*, ACE '07, pages 81–87, Darlinghurst, Australia, Australia, 2007. Australian Computer Society, Inc.

[65] V. Karavirta, J. Helminen, and P. Ihantola. A Mobile Learning Application for Parsons Problems with Automatic Feedback. In *Proceedings of the 12th Koli Calling International Conference on Computing Education Research*, Koli Calling '12, pages 11–18, New York, NY, USA, 2012. ACM.

[66] V. Karavirta, A. Korhonen, and L. Malmi. On the use of resubmissions in automatic assessment systems. *Computer Science Education*, 16(3):229–240, Sept. 2006.

[67] J. Kasurinen and U. Nikula. Estimating Programming Knowledge with Bayesian Knowledge Tracing. In *Proceedings of the 14th Annual ACM SIGCSE Conference on Innovation and Technology in Computer Science Education*, ITiCSE '09, pages 313–317, New York, NY, USA, 2009. ACM.

[68] U. Kiesmueller, S. Sossalla, T. Brinda, and K. Riedhammer. Online Identification of Learner Problem Solving Strategies Using Pattern Recognition Methods. In *Proceedings of the Fifteenth Annual Conference on Innovation and Technology in Computer Science Education*, ITiCSE '10, pages 274–278, New York, NY, USA, 2010. ACM.

[69] U. Kiesmüller. Diagnosing Learners' Problem Solving Strategies Using Learning Environments with Algorithmic Problems in Secondary Education. In *Proceedings of the 8th International Conference on Computing Education Research*, Koli '08, pages 16–24, New York, NY, USA, 2008. ACM.

[70] B. Kitchenham and S. Charters. Guidelines for performing systematic literature reviews in software engineering, 2007.

[71] I. Koprinska, J. Stretton, and K. Yacef. Students at Risk: Detection and Remediation. In *Educational Data Mining*, 2015.

[72] A. Korhonen, L. Malmi, and P. Silvasti. TRAKLA2: a framework for automatically assessed visual algorithm simulation exercises. 2003.

[73] A. N. Kumar. Generation of Problems, Answers, Grade, and Feedback—case Study of a Fully Automated Tutor. *J. Educ. Resour. Comput.*, 5(3), Sept. 2005.

[74] T. Lehtonen. Javala - addictive e-learning of the java programming language. In *Proceedings of Kolin Kolistelut/Koli Calling–Fifth Annual Baltic Conference on Computer Science Education. Joensuu, Finland*, pages 41–48, 2005.

[75] R. Lister. CS Education Research: The Naughties in CSEd Research: A Retrospective. *ACM Inroads*, 1(1):22–24, Mar. 2010.

[76] K. Longi, J. Leinonen, H. Nygren, J. Salmi, A. Klami, and A. Vihavainen. Identification of programmers from typing patterns. In *Proceedings of the 15th Koli Calling Conference on Computing Education Research*, Koli Calling '15, pages 60–67, New York, NY, USA, 2015. ACM.

[77] Ma, R. S. Baker, M. C. Jadud, A. Christine, T. Dy, M. Beatriz, S. Ann, S. A. M. S. Pascua, J. O. Sugay, and E. S. Tabanao. Affective and Behavioral Predictors of Novice Programmer Achievement. In *Proceedings of the 14th Annual ACM SIGCSE Conference on Innovation and Technology in Computer Science Education*, ITiCSE '09, pages 156–160, New York, NY, USA, 2009. ACM.

[78] Ma and Ryan. Coarse-grained Detection of Student Frustration in an Introductory Programming Course. In *Proceedings of the Fifth International Workshop on Computing Education Research Workshop*, ICER '09, pages 75–80, New York, NY, USA, 2009. ACM.

[79] L. Malmi, J. Sheard, Simon, R. Bednarik, J. Helminen, A. Korhonen, N. Myller, J. Sorva, and A. Taherkhani. Characterizing Research in Computing Education: A Preliminary Analysis of the Literature. In *Proceedings of the Sixth International Workshop on Computing Education Research*, ICER '10, pages 3–12, New York, NY, USA, 2010. ACM.

[80] Y. Matsuzawa, K. Okada, and S. Sakai. Programming Process Visualizer: A Proposal of the Tool for Students to Observe Their Programming Process. In *Proceedings of the 18th ACM Conference on Innovation and Technology in Computer Science Education*, ITiCSE '13, pages 46–51, New York, NY, USA, 2013. ACM.

[81] M. McCracken, V. Almstrum, D. Diaz, M. Guzdial, D. Hagan, Y. B.-D. Kolikant, C. Laxer, L. Thomas, I. Utting, and T. Wilusz. A multi-national, multi-institutional study of assessment of programming skills of first-year cs students. In *Working Group Reports*

from ITiCSE on Innovation and Technology in Computer Science Education, ITiCSE-WGR '01, pages 125–180, New York, NY, USA, 2001. ACM.

[82] C. F. Medina, J. R. P. Pérez, V. M. Álvarez Garc'ıa, and del Puerto Paule Ruiz. Assistance in Computer Programming Learning Using Educational Data Mining and Learning Analytics. In *Proceedings of the 18th ACM Conference on Innovation and Technology in Computer Science Education*, ITiCSE '13, pages 237–242, New York, NY, USA, 2013. ACM.

[83] C. Norris, F. Barry, J. B. Fenwick, K. Reid, and J. Rountree. ClockIt: Collecting Quantitative Data on How Beginning Software Developers Really Work. In *Proceedings of the 13th Annual Conference on Innovation and Technology in Computer Science Education*, ITiCSE '08, pages 37–41, New York, NY, USA, 2008. ACM.

[84] R. C. Oliver. How Can Software Metrics Help Novice Programmers? In *Proceedings of the Thirteenth Australasian Computing Education Conference - Volume 114*, ACE '11, pages 55–62, Darlinghurst, Australia, Australia, 2011. Australian Computer Society, Inc.

[85] A. Papancea, J. Spacco, and D. Hovemeyer. An Open Platform for Managing Short Programming Exercises. In *Proceedings of the Ninth Annual International ACM Conference on International Computing Education Research*, ICER '13, pages 47–52, New York, NY, USA, 2013. ACM.

[86] M. Pärtel, M. Luukkainen, A. Vihavainen, and T. Vikberg. Test My Code. *Int. J. Technol. Enhanc. Learn.*, 5(3/4):271–283, Feb. 2013.

[87] A. Petersen, J. Spacco, and A. Vihavainen. An Exploration of Error Quotient in Multiple Contexts. In *Proceedings of the 15th Koli Calling International Conference on Computing Education Research*, Koli Calling '15, New York, NY, USA, 2015. ACM.

[88] R. Pettit, J. Homer, R. Gee, S. Mengel, and A. Starbuck. An Empirical Study of Iterative Improvement in Programming Assignments. In *Proceedings of the 46th ACM Technical Symposium on Computer Science Education*, SIGCSE '15, pages 410–415, New York, NY, USA, 2015. ACM.

[89] M. Piccioni, C. Estler, and B. Meyer. SPOC-supported Introduction to Programming. In *Proceedings of the 2014 Conference on Innovation & Technology in Computer Science Education*, ITiCSE '14, pages 3–8, New York, NY, USA, 2014. ACM.

[90] C. Piech, M. Sahami, J. Huang, and L. Guibas. Autonomously Generating Hints by Inferring Problem Solving Policies. In *Proceedings of the Second (2015) ACM Conference on Learning @ Scale*, L@S '15, pages 195–204, New York, NY, USA, 2015. ACM.

[91] C. Piech, M. Sahami, D. Koller, S. Cooper, and P. Blikstein. Modeling How Students Learn to Program. In *Proceedings of the 43rd ACM Technical Symposium on Computer Science Education*, SIGCSE '12, pages 153–160, New York, NY, USA, 2012. ACM.

[92] A. L. Reilly, P. Denny, D. Kirk, E. Tempero, and S. Y. Yu. On the Differences Between Correct Student Solutions. In *Proceedings of the 18th ACM Conference on Innovation and Technology in Computer Science Education*, ITiCSE '13, pages 177–182, New York, NY, USA, 2013. ACM.

[93] P. J. Robinson. MyPyTutor: An Interactive Tutorial System for Python. In *Proceedings of the 13th Australasian Computing Education Conference – Volume 114*, ACE '11, pages 155–160, Darlinghurst, Australia, Australia, 2011. Australian Computer Society, Inc.

[94] M. M. Rodrigo, E. Tabanao, M. B. Lahoz, and M. C. Jadud. Analyzing online protocols to characterize novice java programmers. *Philippine Journal of Science*, 138(2):177–190, 2009.

[95] P. Runeson and M. Höst. Guidelines for conducting and reporting case study research in software engineering. *Empirical Software Engineering*, 14(2):131–164, Apr. 2009.

[96] M. Satratzemi, V. Dagdilelis, and G. Evagelidis. A System for Program Visualization and Problem-solving Path Assessment of Novice Programmers. In *Proceedings of the 6th Annual Conference on Innovation and Technology in Computer Science Education*, ITiCSE '01, pages 137–140, New York, NY, USA, 2001. ACM.

[97] K. Sharma, P. Jermann, and P. Dillenbourg. Identifying Styles and Paths toward Success in MOOCs. In *Educational Data Mining*, 2015.

[98] T. Sirkiä and J. Sorva. Exploring Programming Misconceptions: An Analysis of Student Mistakes in Visual Program Simulation Exercises. In *Proceedings of the 12th Koli Calling International Conference on Computing Education Research*, Koli Calling '12, pages 19–28, New York, NY, USA, 2012. ACM.

[99] B. Skupas and V. Dagiene. Observations from Semi-automatic Testing of Program Codes in the High School Student Maturity Exam. In *Proceedings of the 10th Koli Calling International Conference on Computing Education Research*, Koli Calling '10, pages 31–36, New York, NY, USA, 2010. ACM.

[100] J. Sorva and T. Sirkiä. UUhistle: A Software Tool for Visual Program Simulation. In *Proceedings of the 10th Koli Calling International Conference on Computing Education Research*, Koli Calling '10, pages 49–54, New York, NY, USA, 2010. ACM.

[101] J. Spacco, P. Denny, B. Richards, D. Babcock, D. Hovemeyer, J. Moscola, and R. Duvall. Analyzing Student Work Patterns Using Programming Exercise Data. In *Proceedings of the 46th ACM Technical Symposium on Computer Science Education*, SIGCSE '15, pages 18–23, New York, NY, USA, 2015. ACM.

[102] J. Spacco, D. Fossati, J. Stamper, and K. Rivers. Towards Improving Programming Habits to Create Better Computer Science Course Outcomes. In *Proceedings of the 18th ACM Conference on Innovation and Technology in Computer Science Education*, ITiCSE '13, pages 243–248, New York, NY, USA, 2013. ACM.

[103] J. Spacco, D. Hovemeyer, W. Pugh, F. Emad, J. K. Hollingsworth, and N. P. Perez. Experiences with Marmoset: Designing and Using an Advanced Submission and Testing System for Programming Courses. *SIGCSE Bull.*, 38(3):13–17, June 2006.

[104] E. S. Tabanao, Ma, and M. C. Jadud. Predicting At-risk Novice Java Programmers Through the Analysis of Online Protocols. In *Proceedings of the Seventh International Workshop on Computing Education Research*, ICER '11, pages 85–92, New York, NY, USA, 2011. ACM.

[105] A. Taherkhani, A. Korhonen, and L. Malmi. Categorizing variations of student-implemented sorting algorithms. *Computer Science Education*, 22(2):109–138, June 2012.

[106] A. Taherkhani and L. Malmi. Beacon-and schema-based method for recognizing algorithms from students' source code. *Journal of Educational Data Mining*, 5(2):69–101, 2013.

[107] R. C. Thomas, A. Karahasanovic, and G. E. Kennedy. An Investigation into Keystroke Latency Metrics As an Indicator of Programming Performance. In *Proceedings of the 7th Australasian Conference on Computing Education - Volume 42*, ACE '05, pages 127–134, Darlinghurst, Australia, Australia, 2005. Australian Computer Society, Inc.

[108] I. Utting, A. E. Tew, M. McCracken, L. Thomas, D. Bouvier, R. Frye, J. Paterson, M. Caspersen, Y. B.-D. Kolikant, J. Sorva, and T. Wilusz. A fresh look at novice programmers' performance and their teachers' expectations. In *Proceedings of the ITiCSE Working Group Reports Conference on Innovation and Technology in Computer Science Education-working Group Reports*, ITiCSE -WGR '13, pages 15–32, New York, NY, USA, 2013. ACM.

[109] A. Vihavainen, J. Helminen, and P. Ihantola. How Novices Tackle Their First Lines of Code in an IDE: Analysis of Programming Session Traces. In *Proceedings of the 14th Koli Calling International Conference on Computing Education Research*, Koli Calling '14, pages 109–116, New York, NY, USA, 2014. ACM.

[110] A. Vihavainen, M. Luukkainen, and P. Ihantola. Analysis of source code snapshot granularity levels. In *Proceedings of the 15th Annual Conference on Information Technology Education*, SIGITE '14, pages 21–26, New York, NY, USA, 2014. ACM.

[111] A. Vihavainen, M. Luukkainen, and J. Kurhila. Using students' programming behavior to predict success in an introductory mathematics course. In *Educational Data Mining*, pages 300–303, 2013.

[112] C. Watson, F. W. B. Li, and J. L. Godwin. No Tests Required: Comparing Traditional and Dynamic Predictors of Programming Success. In *Proceedings of the 45th ACM Technical Symposium on Computer Science Education*, SIGCSE '14, pages 469–474, New York, NY, USA, 2014. ACM.

[113] L. Werner, C. McDowell, and J. Denner. A First Step in Learning Analytics: Pre-processing Low-Level Alice Logging Data of Middle School Students. *Journal of Educational Data Mining*, 5(2), 2013.

[114] L. Werner, C. McDowell, and J. Denner. A first step in learning analytics: pre-processing low-level Alice logging data of middle school students. *JEDM-Journal of Educational Data Mining*, 5(2):11–37, 2013.

[115] L. Werner, C. McDowell, and J. Denner. Middle School Students Using Alice: What Can We Learn from Logging Data? In *Proceeding of the 44th ACM Technical Symposium on Computer Science Education*, SIGCSE '13, pages 507–512, New York, NY, USA, 2013. ACM.

[116] T. Winters and T. Payne. What Do Students Know?: An Outcomes-based Assessment System. In *Proceedings of the First International Workshop on Computing Education Research*, ICER '05, pages 165–172, New York, NY, USA, 2005. ACM.

[117] M. Yudelson, R. Hosseini, A. Vihavainen, and P. Brusilovsky. Investigating automated student modeling in a Java MOOC. In *Educational Data Mining*, pages 261–264, 2014.

[118] D. Zingaro, Y. Cherenkova, O. Karpova, and A. Petersen. Facilitating Code-writing in PI Classes. In *Proceedings of the 44th ACM Technical Symposium on Computer Science Education*, SIGCSE '13, pages 585–590, New York, NY, USA, 2013. ACM.

APPENDIX

A. DATA EXTRACTION FORM

Table 6: Categories and questions/fields in the data extraction form.

Paper Identification	
Paper title Venue	
Motivation / Research Goal	
Research goal (copy/paste verbatim from the source if possible) Are research questions explicitly stated?	. . . * Yes/No
Methodological Aspects / Approach	
Single or multi-context Single or longitudinal Type of research Replication study Name of replicated study Does the paper use a theory or known model (the theory/model must be actually used somehow in the research)	Multiple choice: single/many/no course, single/multi/no institution Single choice: single or longitudinal Multiple choice: case study/constructive research/experiment/study/survey research/other(please provide details) Single choice: Yes/No/Partial . . . Yes/No
Context of the Research	
Programming language IDE Course related setting Name of course	. . . . . . Single choice: within a course/not-course setting/Unknown . . .
Subjects	
Number of students Level of students/course (e.g., third year undergraduate) Majors/non-majors/mixed	. . . . . . Single choice: majors/non-majors/mixed/unknown/other(please provide details)
Task	
Programming problem (brief description) Single or multiple tasks Open-ended task Tasks are graded Automated feedback	. . . Single choice: single/multiple Yes/No Yes/No/Unknown Yes/No/Unknown
Data that is Collected	
Type of data (e.g., coding snapshots, compiler errors, program output, time-stamps, exam results, demographics, eye movements, etc.) Frequency of data collection Anonymized data Data analyzed without informing students Open dataset	. . . . . . Yes/No/Unknown Yes/No/Unknown Yes/No/Unknown
Research Instruments (how is the data collected)	
IDE or computer instrumentation (only looking at submissions => no instrumentation) Name of the data collection system Other data collection instruments (e.g., questionnaire, eye tracking etc.)	Yes/No/Unknown . . . . . .
Analysis Methods (how is the data analyzed)	
Statistics	Multiple choice: descriptive statistics/exploratory analysis (correlation, regression, factor analysis, etc.)/inferential analysis (Bayesian, t-test, etc.)/interpretive classification (based on existing classification scheme or one that is refined during the analysis)/interpretive qualitative analysis (data-driven formation of qualitatively different categories)/automated classification/other(please provide details)
Results	

Continued on next page

Table 6 – *Continued from previous page*

How are the results used or proposed to be used (e.g., intervention, personalization, course review, understanding student behavior)	. . .
Name of the data collection system	. . .
Ethical issues are discussed	Yes/No
Privacy issues are discussed	Yes/No
Quality Assessment (check all that apply)	
The hypotheses/aims/objectives of the study are clearly described.	
The choices of participants, instruments and methods are clearly motivated to address the research questions.	
The characteristics of the students included in the study are clearly described.	
The data collected are sufficient for the purpose of the study	
The research process is clearly described.	
The main findings of the study are clearly described.	
A chain of evidence from observations to conclusions is clearly established.	
Actual probability and significant values have been reported.	
An attempt was made to reduce bias.	
Appropriate analysis procedures were used to assess the main outcomes.	
Appropriate measures are used to ensure accurate, valid and reliable outcomes.	
Threats to validity are analyzed in a systematic way and countermeasures were taken to reduce threats.	
Confounding factors have been acknowledged and discussed.	
Ethical issues were acknowledged and discussed.	
Conclusions, implications for practice, and future research were suitably reported.	

*Free form text

B. LIST OF PRIMARY PAPERS

Title	**Year**	**Citation**
A System for Program Visualization and Problem-Solving Path Assessment of Novice Programmer	2001	[96]
A First Look at Novice CompilationBehaviour Using BlueJ	2005	[60]
An analysis of patterns of debugging among novice computer science students	2005	[6]
An Investigation into Keystroke Latency Metrics As an Indicator of Programming Performance	2005	[107]
High School Programming - A Beyond-Syntax Analysis of Novice Programmers' Difficulties	2005	[49]
Automated assessment and experiences of teaching programming.	2005	[55]
Javala - Addictive E-Learning of the Java Programming Language	2005	[74]
Experiences with marmoset: designing and using an advanced submission and testing system for programming courses	2006	[103]
Methods and tools for exploring novice compilation behaviour	2006	[61]
On the use of resubmissions in automatic assessment systems	2006	[66]
Program Annotations: Feedback For Students Learning To Program	2006	[12]
Difficulties Experienced by Students in Maintaining Object Oriented Systems	2007	[64]
Which pointer errors do students make?	2007	[5]
ClockIt: collecting quantitative data on how beginning software developers really work	2008	[83]
Diagnosing learners' problem solving strategies using learning environments with algorithmic problems in secondary education	2008	[69]
Mining Data from an Automated Grading and Testing System by Adding Rich Reporting Capabilities	2008	[9]
Affective and behavioral predictors of novice programmer achievement	2009	[77]
Another Look at the Behaviors of Novice Programmers	2009	[46]
Coarse-grained detection of student frustration in an introductory programming course	2009	[78]
Comparing effective and ineffective behaviors of student programmers	2009	[43]
Dereferee: exploring pointer mismanagement in student code	2009	[8]
Estimating programming knowledge with Bayesian knowledge tracing	2009	[67]
A Detector for Non-literal Java Errors	2010	[36]
Data Mining for Individualised Hints in eLearning	2010	[33]
Observations from semi-automatic testing of program codes in the high school student maturity exam	2010	[99]
Online Identification of Learner Problem Solving Strategies Using Pattern Recognition Methods	2010	[68]
String formatting considered harmful for novice programmers	2010	[58]
CodeWrite: Supporting Student-Driven Practice of Java	2011	[31]
How can software metrics help novice programmers?	2011	[84]
MyPyTutor: an interactive tutorial system for Python	2011	[93]

Predicting at-risk novice Java programmers through the analysis of online protocols	2011	[104]
Understanding the Syntax Barrier for Novices	2011	[32]
Which aspects of novice programmers' usage of an IDE predict learning outcomes	2011	[37]
A fast measure for identifying at-risk students in computer science	2012	[45]
All syntax errors are not equal	2012	[30]
Categorizing variations of student-implemented sorting algorithms	2012	[105]
Exploring influences on student adherence to test-driven development	2012	[20]
Exploring programming misconceptions: an analysis of student mistakes in visual program simulation exercises	2012	[98]
How do students solve parsons programming problems?: an analysis of interaction traces	2012	[53]
Implementation of a Smart Lab for Teachers of Novice Programming	2012	[7]
Modeling how students learn to program	2012	[91]
Running students' software tests against each others' code: new life for an old ”gimmick”	2012	[41]
A First Step in Learning Analytics: Pre-processing Low-Level Alice Logging Data of Middle School Students	2013	[114]
Assistance in computer programming learning using educational data mining and learning analytics	2013	[82]
Beacon- and Schema-Based Method for Recognizing Algorithms from Students Source Code	2013	[106]
Effective and Ineffective Software Testing Behaviors by Novice Programmers	2013	[21]
Impacts of adaptive feedback on teaching test-driven development	2013	[22]
Middle school students using Alice: what can we learn from logging data?	2013	[115]
On the differences between correct student solutions	2013	[92]
Programming process visualizer: a proposal of the tool for students to observe their programming process	2013	[80]
Recording and Analyzing In-Browser Programming Sessions	2013	[52]
Towards improving programming habits to create better computer science course outcomes	2013	[102]
Using Students' Programming Behavior to Predict Success in an Introductory Mathematics Course	2013	[111]
A formative study of influences on student testing behaviors	2014	[23]
Adaptively identifying non-terminating code when testing student programs	2014	[42]
Blackbox: A Large Scale Repository of Novice ProgrammersâĂŹ Activity	2014	[18]
Do student programmers all tend to write the same software tests?	2014	[40]
Enhancing syntax error messages appears ineffectual	2014	[29]
Exploring Problem Solving Paths in a Java Programming Course	2014	[56]
How (not) to introduce badges to online exercises	2014	[50]
How novices tackle their first lines of code in an IDE: analysis of programming session traces	2014	[109]
Identifying challenging CS1 concepts in a large problem dataset	2014	[27]
Increasing the effectiveness of automated assessment by increasing marking granularity and feedback units	2014	[44]
Investigating Automated Student Modeling in a Java MOOC	2014	[117]
Investigating novice programming mistakes: educator beliefs vs. student data	2014	[17]
No tests required: comparing traditional and dynamic predictors of programming success	2014	[112]
Responses to adaptive feedback for software testing	2014	[24]
SPOC-supported introduction to programming	2014	[89]
37 Million Compilations: Investigating Novice Programming Mistakes in Large-Scale Student Data	2015	[10]
An Empirical Study of Iterative Improvement in Programming Assignments	2015	[88]
An Incremental Hint System For Automated Programming Assignments	2015	[11]
Analyzing Student Work Patterns Using Programming Exercise Data	2015	[101]
Evaluation of Source Code with Item Response Theory	2015	[13]
Identifying Styles and Paths toward Success in MOOCs	2015	[97]
Students at Risk: Detection and Remediation	2015	[71]
Towards Incremental Separation of Surmountable and Insurmountable Programming Difficulties	2015	[25]