Learning Programming, Syntax Errors and Institution-specific Factors

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

Raymond Lister
University of Technology
Sydney, Australia
Raymond.Lister@uts.edu.au

Shahil Lal
University of Sydney
Australia
Shahil.Lal@uts.edu.au

Arto Hellas
University of Helsinki
Helsinki, Finland
avihavai@cs.helsinki.fi

ABSTRACT

Learning programming is a road that is paved with mistakes. Initially, novices are bound to write code with syntactic mistakes, but after a while semantic mistakes take a larger role in the novice programmers' lives.

Researchers who wish to understand that road are increasingly using data recorded from students' programming processes. Such data can be used to draw inferences on the typical errors, and on how students approach fixing them. At the same time, if the lens that is used to analyze such data is used only from one angle, the view is likely to be narrow.

In this work, we replicate a previous multi-institutional study by Brown et al. [5]. That study used a large scale programming process data repository to analyze mistakes that novices make while learning programming. In our single institution replication of that study, we use data collected from approximately 800 students. We investigate the frequency, time required to fix, and the development of mistakes through the semester. We contrast our findings from our single institution with the multi-institutional study, and show that whilst the data collection tools and the research methodology are the same, the results can differ solely due to how the course is conducted.

CCS CONCEPTS

• **Social and professional topics → Computer science education**;

KEYWORDS

programming, replication, syntactic errors, semantic errors, student mistakes

ACE 2018, January 30-February 2, 2018,, Brisbane, QLD, Australia
© 2018 Association for Computing Machinery.
ACM ISBN 978-1-4503-6340-2/18/01...$15.00
https://doi.org/10.1145/3160489.3160490.

1 INTRODUCTION

The emergence of tools that gather data from students' programming process has led to a stream of research that analyzes such data with the hope of gaining insight into how students learn to program [12]. State of the art studies focus on the analysis of large repositories. One such study used the Blackbox data set [6] that gathers data from large numbers of individuals, world-wide. Whilst such data can provide insight on problems that students face when learning programming on a global level, the data may also obscure characteristics that vary across the institutions that contribute to the global data repositories.

There is a need to balance studies using global data with studies from individual institutions, so that the impact of factors that vary between institutions can be assessed. Such factors include the level of prior education, expected course outcomes, course level, and more. Some institutions may have students' working on tens of weekly assignments and expect that students invest tens of hours of work on learning programming each week, whilst other institutions may only seek to provide a glance at what programming could be like, and have students work on only an assignment or two per week – if at all. A systematic study of such institution-specific characteristics, through studies that replicate previous research, may lead to accumulated understanding of the factors that contribute to students' struggles and learning outcomes.

In this work, we replicate the global study [5] by Brown et al. in a single institutional context and study whether the results from a single institution differ from the results drawn from many institutions around the world.

As the programming environment and the methodology used to extract errors are the same as in the original study, our study seeks to determine whether the way a course is conducted at a specific institution matters. Our goal is to both emphasize the need for replication studies – as recently emphasized by Ihantola et al. [11] and Ahadi et al. [1] – and to call for a stream of research that seeks to determine factors that contribute to institutional differences.

Our main research question in this article is: *How different are the syntactic and semantic errors observed at a single institution to those that have been observed in analysis of combined student populations from around the world*? With this question, we determine to what extent results from generic data sets, such as the Blackbox project or MOOCs, generalize to single institutions. This topic has been previously explored in the context of student modeling, where

Yudelson et al. [22] observed that single courses may offer better generalizability than larger datasets.

This article is organized as follows. Next, in Section 2, we review the recent work on syntax errors in learning programming, and point out the recent emphasis on replication studies. This is followed by a description of the research method in Section 3. The results of the study are presented and discussed in Section 4, which is followed conclusions.

2 BACKGROUND

2.1 Syntax Errors and Programming

Denny et al. [10] conducted a study on students who are working on Java programs in an online programming environment. In their study, most of the submissions to the online system included code with syntax errors, leading to the conclusion that learning the syntax in Java is not straightforward. In a subsequent study, Denny et al. [9] studied how frequent specific syntax errors are, and observed that errors have varying difficulty levels.

This line of work was later continued by Vihavainen et al. [18], who studied the frequency of syntax errors as novices work on programming assignments in an IDE. Contrary to the results of Denny et al. [9, 10], they observed that when novices were programming in an IDE that highlighted the syntax errors and provided feedback on how to fix them, over 90% of the submissions were syntactically correct. They suggested that some of the previous results that are related to syntax errors are a product of the working environment: if students receive feedback on the syntactic correctness of their code only during submission, submissions will be used to assess the syntactic correctness of code.

In addition to the programming environment used, the programming language itself may also influence the problems that students face. Stefik and Siebert [16] conducted reading surveys to study the perceived intuitiveness of the syntax of various programming languages. They found that languages that are closer to the English language may be more intuitive to novices than those languages that follow a more traditional syntax. As the study was conducted on an English speaking population, it is a good question whether this result would hold if the participants did not have English as their primary or secondary language. The understandability of the programming language has raised interest in other contexts as well. For example, Miller [14] studied student challenges in a more complex environment that included the teaching of object oriented programming, and identified referencing as a source of multiple errors. Altadmri and Brown [2] analyzed the frequency, time-to-fix, and spread of errors among users and showed how these factors inter-relate over the duration of the first year of programming.

2.2 Syntax Errors and Success

Syntax errors that students face as they are programming have been used to create metrics of student's coding success. In 2006, Jadud introduced an algorithm called Error Quotient (EQ), which analyses successive states in student's programming process and provides a metric on how well the student has been able to fix any encountered syntax errors [13]. The EQ algorithm was later extended by Watson et al. to include information on the amount of time that the student takes to fix an error; the Watwin algorithm [20]. Other methods for assessing students' success based on their errors in the programming process have been proposed more recently, for example the Normalized Programming State Model [7], and the Repeated Error Density [4].

In a recent study, Petersen et al. [15] studied the applicability of the EQ algorithm in different contexts. They varied the programming language and the programming environment, and observed that the performance of the EQ algorithm was not as high outside the original EQ context. They concluded that there is a need for further studies, and highlighted the need for replication studies.

A few studies exist where the syntax error messages have been modified to improve their understandability. Both Watson et al. [21] and Becker [3] have suggested that improved syntax errors messages could improve students' performance, but Denny [8] noted that enhanced syntax error messages do not always work. Here, as before, it is likely that differences in these research results are due to differences in institutional contexts.

2.3 Importance of the context and replication

Whilst the previous studies have mostly been conducted in a single institution context, a few studies highlighted the importance of the teaching context, including the language, tools, and practices [15, 16, 18].

In light of the relatively high failure rates in introductory programming courses [19], it is meaningful to point out that how the course is taught is important. If the students struggle with syntax, the teachers may take steps to remedy the issue; a survey by Vihavainen et al. [17] pointed out that a simple teaching intervention in an introductory programming class improved the overall pass rate, on average, by one third.

At the same time, most of the results analyzed in the survey were positive, and it is possible that educators have not published results from non-functioning teaching interventions. Such an observation would be in line with the results of a recent study on attitudes towards replication, where one of the observations was that novel results are more valuable than confirming old studies [1]. In general, whilst we cannot go so far as to say that there is a replication crisis in computing education research, there is a need for replicating previous studies to increase the reliability and validity of the results. This statement is in line with other recent works such as a recent ITiCSE working group [12], who stated that *[one of the Grand Challenges to improve the field of Computing Education Research] is to systematically analyze and verify previous studies using data from multiple contexts to tease out factors that contribute to previously observed outcomes.*

3 METHOD

3.1 Data and Context

The data for this study comes from two semesters of an introductory programming course organized at the University of Technology, Sydney (UTS). The data contains programming process recordings from 800 students, who generated in total ≈1.5 million distinct events from ≈13,000 programming sessions. The language taught is Java, and covers common programming concepts including variables, basic I/O, methods, conditionals, loops, arrays, elementary search algorithms and objects. Students complete practical programming assessments in two hour weekly laboratory sessions. The first hour of each laboratory session is dedicated to assessment followed by the second hour which has the form of a normal tutorial. During the first hour of the laboratory, students work under exam condition on a series of mastery tests. A total number of 22 compulsory mastery tests need to be successfully passed by students over 12 weeks.

Students work at their own pace. Successful completion of each assessment item by a student unlocks the next assessment item. There are no limitations on the number of attempts/sessions to complete the exercises. During the first hour of the laboratory, students student access to the internet is almost entirely prevented, along with access to their own file space. The only website which is accessible is the website from which students download the prepared skeleton code for the assessment item that they are about to attempt. The skeleton code of each assessment item provides students with enough abstracted detail about the exercise's code, its output and the required manipulations to complete the code. This website is used for automatic assessment of students' work in the course and is always accessible for students so they can practice the lab tests as much as they like prior to taking the test under exam conditions. Thus when a student attempts a lab test it is likely that the student had already seen the test prior to the session. The VMware running on the PCs in the laboratory is designed so that no two students seating next to each other can work simultaneously on the same test.

When a student downloads the skeleton code for an exercise, BlueJ is automatically opened and the downloaded project is loaded. Encrypted information about the student as well as the name of the lab test are assigned to the experiment identifier and participant identifier variables of BlueJ. Those two identifier variables are required by Blackbox [6] when each Java code snapshot is uploaded to Blackbox. While attempting a test, students may at any time upload their source code back to the website from where they downloaded the test. There is no limitation in the number of submissions to the assessment website, and no grading penalty for multiple submissions. Each submission is graded using a small number of test cases.

The majority of the students participating in this replication study are studying full time, speak English, are 18-19 years old, and approximately 80% of them are male. About half of the students have no previous exposure to any programming languages, whilst approximately 15% of the students claimed to have had at least some prior experience with Java.

3.2 Student Mistakes

Brown et al.'s study [5] reviewed a total number of 18 mistakes. These mistakes are summarized in Table 1.

As in the original study, in this replication each source file is tracked over time: At each compilation, the source file is checked for the above mentioned eighteen errors. If any mistake is present, then the source file at the next compilation is checked to see if the mistake is no longer present. If the mistake is no longer found, it is counted as one instance of the mistake. Further occurrences in the same source file are regarded as further instances.

Table 1: Student Mistakes

Shorthand	Explanation
Syntax errors:	
A	Confusing = with ==
C	Mismatched parentheses
D	Confusing & with &&
E	Spurious semi-colon after `if`, `for`, `while`
F	Wrong separator in `for`
G	Wrong brackets in `if`
H	Using reserved keywords
J	Forgetting parentheses when calling methods
K	Spurious semi-colon after method header
L	Less-than / greater-than operators wrong
P	Including types in actual method arguments
Type errors:	
I	Calling method with wrong types
Q	Type mismatch when assigning method result
Other semantic errors:	
B	Using == to compare strings
M	Invoking instance method as static
N	Discarding method return
O	Missing return statement
R	Missing methods when implementing interface

According to the original study, the authors calculated the time in seconds between the first appearance of the mistake and the possible fix, with a ceiling of 1000 seconds (just over 15 minutes). Any mistake that takes longer than 1000 seconds to fix is considered to have never been fixed.

3.3 Contextual Differences

Some things about the participating novice programmers are better known in this replication than in the original study [5]. For example, the students in the replication study are known to be from an introductory programming class. More importantly, the students in our study are working under exam conditions, where receiving help from other students or consulting other resources is forbidden. The start and finish times of lab test sessions are well defined, so we can extract and study student activity that is known to have occurred under exam conditions. Also, in this study, we know with certainty when the course started and when it ended, whilst in the original study, no certain information on course start and end times (if such existed) was available.

4 RESULTS AND DISCUSSION

4.1 Mistake review

The relative frequency of mistakes in the replication and in the original study are presented in Table 2. The top five mistakes are the same in both studies, but the exact ordering is different. The most frequent mistake in both studies is mistake C (mismatched parentheses) which is a syntactic mistake. The second and third most common mistakes in both studies are semantic mistake O (missing return statement) and type mistake I (calling method with wrong types), although the order of these two mistakes is different in the two studies . The fourth and fifth most common mistakes are the same in both studies (Discarding method return and Confusing = with ==), although once again the order of these two mistakes is different in the two studies .

The rankings of mistakes in the two studies are often quite similar until our sample size for each mistake reaches 33. As the frequency of each mistakes drops in the data collected for our replication, it is to be expected that there would be some large differences in the rankings in the two studies, due to low sample size in our replication.

There exists a set of mistakes that are extremely rare in our data, such as mistake B (using == to compare strings). That specific difference can be explained simply: very few of our lab tests in the replication required students to compare strings.

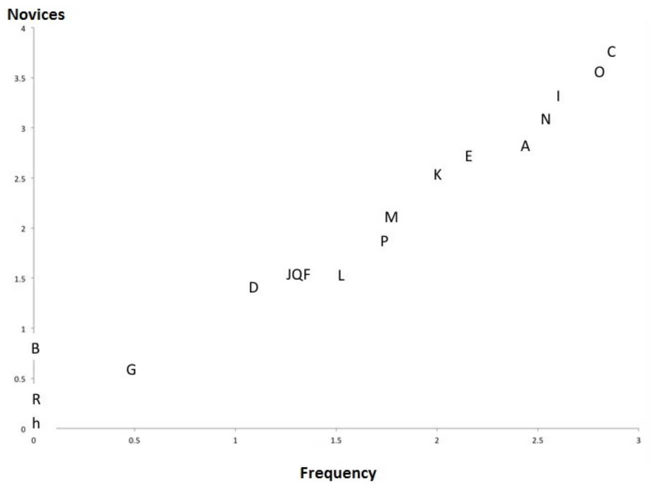
Table 2: Comparison of the relative frequency of different mistakes between the two studies. The second and third columns show respectively the rank of the frequency (i.e. "R1" is the highest) at our institution ("Us") and in the original study ("Orig"). The fourth column shows the distance between the rankings. The final column shows the actual frequency of that error in the replication.

Mistake	*Us*	Orig.	Dist.	Freq.	Type
C	R1	R1	0	715	Syntax
O	R2	R3	1	659	Semantic
I	R3	R2	1	393	Type
N	R4	R5	1	341	Semantic
A	R5	R4	1	262	Syntax
E	R6	R10	4	138	Syntax
K	R7	R11	4	96	Syntax
M	R8	R7	1	56	Semantic
P	R9	R9	0	55	Syntax
L	R10	R15	5	33	Syntax
F	R11	R16	5	21	Syntax
Q	R12	R14	2	19	Type
J	R13	R13	0	18	Syntax
D	R14	R12	2	12	Syntax
G	R15	R18	3	3	Syntax
B	R16	R6	10	1	Semantic
R	R17	R8	9	1	Semantic
H	R18	R17	1	1	Syntax

When analyzing the relationship between the frequency of a mistake and the number of distinct novices who make that mistake (see Fig. 1), a simple relationship can be observed: the mistakes that appear more often are also made by more students.

Of the 18 mistakes reviewed in Figure 2, mistakes B, E and K show above average repeated mistakes in our replication compared to Brown *et al.*. In the case of mistake B (using == to compare strings), and as mentioned before, very few of our lab tests in the replication required students to compare strings, so the high relative repetition in our UTS replication data may be an artifact of small sample size.

Figure 1: Plot of overall frequency of mistakes and the number of novices who made a mistake in the replication (Log10 transformed).



Among mistakes with below average repeats compared to Brown *et al.*, the majority show considerable reduction. Mistake R shows the largest reduction but the reason is simple: the students at our institute are not taught to use interfaces in their introductory programming course. Most mistakes (D, F, G, J, L, P, Q, R) are only made once by students in our institute which shows partial overlap with the data presented in the Brown *et al.* study.

Figure 2: A comparison between the two studies of the average number of times that each user repeated a mistake.

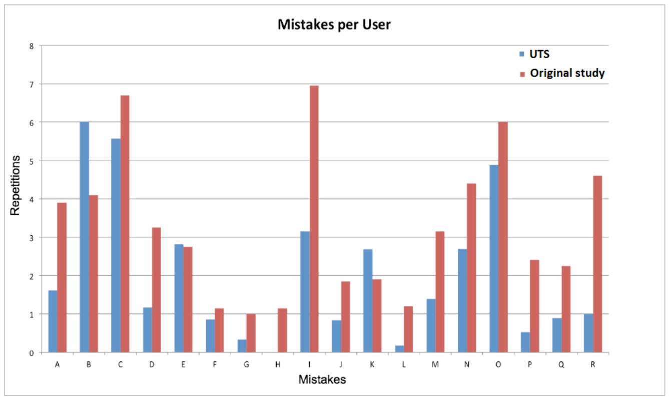


Table 3 shows the median time needed to correct mistakes in both the original study and in our replication. Our students spent the most time (almost 40 seconds) fixing the type error Q (i.e. type mismatch when assigning method result). An incorrect semicolon after a condition (E) is the second mistake which required a long time for our students to fix (35 seconds). However, our students spent far less time fixing those two mistakes Q and E than students spent fixing many other types of mistakes in the original study. In general, there is a significant reduction ($p < 0.01$, t-test) on the average time spent on fixing errors in our data compared to the data of the original study. One potential explanation for this time difference is that the data collected from our institute was collected during a time constrained exam, where students working at their own pace may have only attempted a specific lab test when they had prepared for that test and were confident they would pass that test.

Table 3: Median time (in seconds) needed to fix the given mistakes for our institution (column "Us") and the original study (column "Orig"), and the frequency of the mistakes in our dataset.

Mistake	Us	Orig.	*Freq.*
Syntax errors			
C	12	17	715
A	13	111	262
E	35	401	138
K	11	51	96
P	17	24	55
L	10	12	33
F	13	36	21
J	12	34	18
D	15.5	>1000	12
G	10	26	3
H	10	22	1
Type errors			
I	24	58	393
Q	39.9	115	19
Other semantic errors			
O	15	37	659
N	17	>1000	341
M	11	48	56
B	25	>1000	1
R	19.5	104	1

4.2 Mistake Frequencies Over Course

Brown *et al.* investigated the development of mistakes over time. In their study, they focused on data collected in a year, starting in September and running to August the following year: an approximation to the academic year in the northern hemisphere. In our replication study, we had the advantage of knowing exactly when our teaching periods began and ended.

Figure 4 represents the frequency of three different categories of mistakes over the weeks of the semester at our institute. The beginning of the semester shows a very high peak on the frequency of the syntactic error, which decreases rapidly. The frequency of different types of mistakes for the second half of our semester is consistent with what is presented in the original study. In contrast to the original study, however, students at our institute struggle with semantic mistakes more in the first half of the semester.

Figure 3: Plot of the overall frequency of mistakes and the median time taken to fix those mistakes (Log10 transformed).

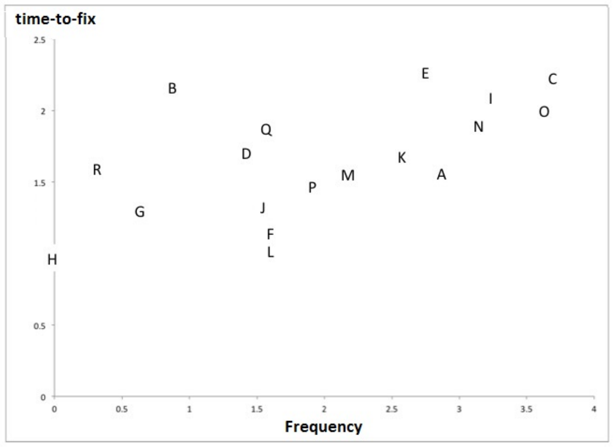
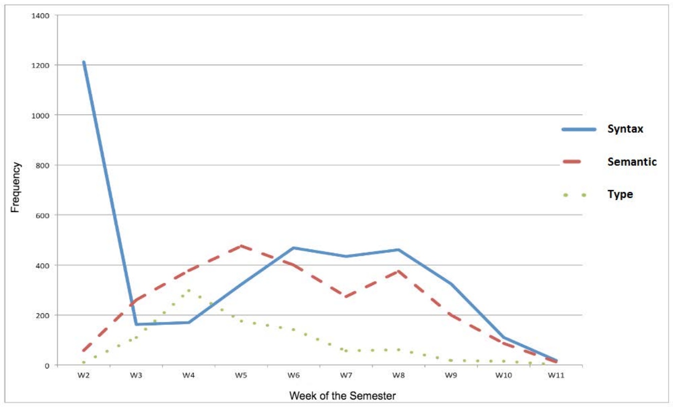


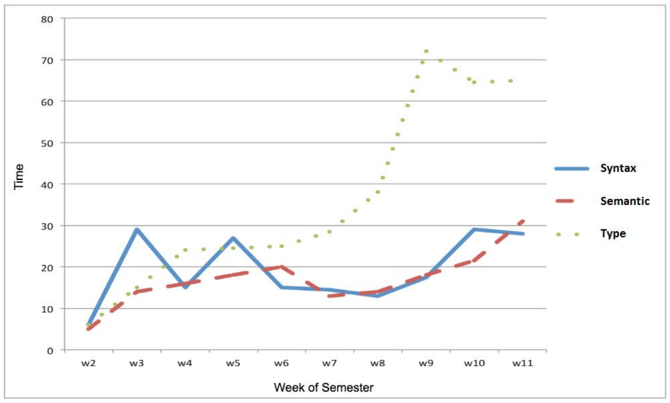
Figure 4: Frequency of mistakes categories over semester in the replication. Week one is not present in the x axis as there are no laboratories running in week one.



In the original study, syntax errors took the least (median) time to fix, semantic mistakes took the most time, with type mistakes typically taking time between syntax and semantic errors. However, in our replication (Figure 5) the type category represents the most time-consuming class of error, across most of the semester. During the early part of the semester, syntactic errors typically take most time to fix. During the second half of the semester, semantic and syntactic mistakes take an approximately equal (median) time to fix. The time to fix of all three types of errors increases toward the end of the semester; especially for type errors. That late increase in all three error types probably reflects the increasing length and sophistication of the lab tests.

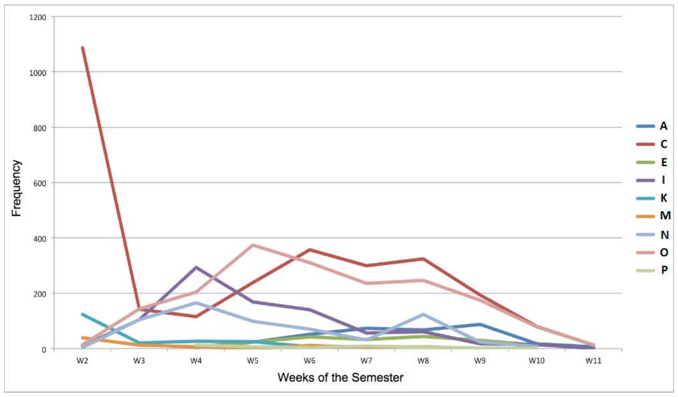
Figure 6 reviews the frequency of different errors during the semester. Errors O and C are for most of the semester a near equally common pair, much more common than the other errors. Errors

Figure 5: Median time needed to fix different types of errors during different weeks of the semester, in the replication.



I and N also form a near equally common pair for most of the semester. Errors A E, K, M are relatively infrequent.

Figure 6: The frequency of different errors during the semester, in the replication. Some errors are not shown because a very small proportion of students made those mistakes.



4.3 Limitations of the work

As explained before, the results in our replication study are mainly generated by the same tool used in the original study. However, sometimes information not included in Brown *et al.* [5] made it difficult to generate the results the same way as it was done in the original study. For example, we were not sure how to normalize the frequency values the same way as the original study as we did not know which compilation events to include as the means of normalization.

We have only analyzed the data generated by our students when they did the lab tests under exam conditions. We did not capture and analyze the programming that the students did to prepare for taking the lab tests. Similarly, the authors of the original study acknowledged that they could not know if they had captured all the programming activity of their participants.

5 CONCLUSIONS

This study utilizes data collected via the Blackbox project from around 800 novice Java programming students at a known educational institution. We have provided results about the distribution of mistakes among those novices and the time required to fix the mistakes. We have compared our results to those of Brown et al. To our knowledge, this study is the first context-specific replication of the original context-free study by Brown et al. [5]. The IDE used to write the Java code, the infrastructure used to collect the data, the tools used to generate the results, and the research method are identical between the original study and the replicated study. However, some of the results from the context studied here differ from the original report.

In general, our results show that the syntactic errors are harder to fix compared to semantic mistakes, when students are working under our institution-specific conditions. In contrast, Brown et al. reported that semantic mistakes were harder. We believe the main reason behind this difference is that the data analyzed by Brown et al. comes from unknown contexts where the expertise level, the environment, the course level and many other factors are likely to be different from our institutional context.

Effectively our results indicate that (1) while the results generated via data collected world-wide from largely unknown particpants sets an important reference point, those results do not necessarily reflect what applies to students at a specific institution, (2) the results in the original study are not solely the product of the tools used, and (3) there a large number of open questions to answer if we wish to understand the institutional factors that contribute to differences between our results and the results of the study that we replicated.

We encourage others to collect and report upon similar data from their institutional contexts. With enough replications, the factors affecting the performance of novices will become more clear.

6 ACKNOWLEDGMENT

We cannot thank enough Neil Brown and Amjad Altadmri, coauthors of the original Brown et al. study, who agreed to execute their customized code on the Blackbox machine for the data collected from our students enrolled at the University of Technology, Sydney. Computing education needs more replication work, and replications are usually difficult without the cooperation of people from the original study. Neil and Amjad set an admirable example for others in computing education research to follow.

REFERENCES

[1] Alireza Ahadi, Arto Hellas, Petri Ihantola, Ari Korhonen, and Andrew Petersen. 2016. Replication in computing education research: researcher attitudes and experiences. In *Proceedings of the 16th Koli Calling International Conference on Computing Education Research*. ACM, 2–11.
[2] Amjad Altadmri and Neil C.C. Brown. 2015. 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)*. ACM, New York, NY, USA, 522–527. https://doi.org/10.1145/2676723.2677258
[3] Brett A. Becker. 2016. An Effective Approach to Enhancing Compiler Error Messages. In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education (SIGCSE '16)*. ACM, New York, NY, USA, 126–131. https://doi.org/10.1145/2839509.2844584
[4] Brett A Becker. 2016. A new metric to quantify repeated compiler errors for novice programmers. In *Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education*. ACM, 296–301.

[5] Neil CC Brown and Amjad Altadmri. 2014. Investigating novice programming mistakes: educator beliefs vs. student data. In *Proceedings of the tenth annual conference on International computing education research*. ACM, 43–50.

[6] Neil Christopher Charles Brown, Michael Kölling, Davin McCall, and Ian Utting. 2014. Blackbox: a large scale repository of novice programmers' activity. In *Proceedings of the 45th ACM technical symposium on Computer science education*. ACM, 223–228.

[7] Adam S. Carter, Christopher D. Hundhausen, and Olusola Adesope. 2015. The Normalized Programming State Model: Predicting Student Performance in Computing Courses Based on Programming Behavior. In *Proceedings of the Eleventh Annual International Conference on International Computing Education Research (ICER '15)*. ACM, New York, NY, USA, 141–150. https://doi.org/10.1145/2787622.2787710

[8] Paul Denny, Andrew Luxton-Reilly, and Dave Carpenter. 2014. Enhancing Syntax Error Messages Appears Ineffectual. In *Proceedings of the 2014 Conference on Innovation & Technology in Computer Science Education (ITiCSE '14)*. ACM, New York, NY, USA, 273–278. https://doi.org/10.1145/2591708.2591748

[9] Paul Denny, Andrew Luxton-Reilly, and Ewan Tempero. 2012. All Syntax Errors Are Not Equal. In *Proceedings of the 17th ACM Annual Conference on Innovation and Technology in Computer Science Education (ITiCSE '12)*. ACM, New York, NY, USA, 75–80. https://doi.org/10.1145/2325296.2325318

[10] Paul Denny, Andrew Luxton-Reilly, Ewan Tempero, and Jacob Hendrickx. 2011. Understanding the Syntax Barrier for Novices. In *Proceedings of the 16th Annual Joint Conference on Innovation and Technology in Computer Science Education (ITiCSE '11)*. ACM, New York, NY, USA, 208–212. https://doi.org/10.1145/1999747.1999807

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

[12] Petri Ihantola, Arto Vihavainen, Alireza Ahadi, Matthew Butler, Jürgen Börstler, Stephen H. Edwards, Essi Isohanni, Ari Korhonen, Andrew Petersen, Kelly Rivers, Miguel Ángel Rubio, Judy Sheard, Bronius Skupas, Jaime Spacco, Claudia Szabo, and Daniel Toll. 2015. Educational Data Mining and Learning Analytics in Programming: Literature Review and Case Studies. In *Proceedings of the 2015 ITiCSE on Working Group Reports (ITICSE-WGR '15)*. ACM, New York, NY, USA, 41–63. https://doi.org/10.1145/2858796.2858798

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

[14] Craig S. Miller and Amber Settle. 2016. Some Trouble with Transparency: An Analysis of Student Errors with Object-oriented Python. In *Proceedings of the 2016 ACM Conference on International Computing Education Research (ICER '16)*. ACM, New York, NY, USA, 133–141. https://doi.org/10.1145/2960310.2960327

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

[16] Andreas Stefik and Susanna Siebert. 2013. An Empirical Investigation into Programming Language Syntax. *Trans. Comput. Educ.* 13, 4, Article 19 (Nov. 2013), 40 pages. https://doi.org/10.1145/2534973

[17] Arto Vihavainen, Jonne Airaksinen, and Christopher Watson. 2014. A Systematic Review of Approaches for Teaching Introductory Programming and Their Influence on Success. In *Proceedings of the Tenth Annual Conference on International Computing Education Research (ICER '14)*. ACM, New York, NY, USA, 19–26. https://doi.org/10.1145/2632320.2632349

[18] Arto Vihavainen, Juha Helminen, and Petri Ihantola. 2014. 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*. ACM, 109–116.

[19] Christopher Watson and Frederick WB Li. 2014. Failure rates in introductory programming revisited. In *Proceedings of the 2014 conference on Innovation & technology in computer science education*. ACM, 39–44.

[20] Christopher Watson, Frederick WB Li, and Jamie L Godwin. 2013. Predicting Performance in an Introductory Programming Course by Logging and Analyzing Student Programming Behavior. In *Advanced Learning Technologies (ICALT), 2013 IEEE 13th International Conference on*. IEEE, 319–323.

[21] Christopher Watson, Frederick W. B. Li, and Jamie L. Godwin. 2012. BlueFix: Using Crowd-sourced Feedback to Support Programming Students in Error Diagnosis and Repair. In *Proceedings of the 11th International Conference on Advances in Web-Based Learning (ICWL'12)*. Springer-Verlag, Berlin, Heidelberg, 228–239. https://doi.org/10.1007/978-3-642-33642-3_25

[22] Michael Yudelson, Steve Fancsali, Steve Ritter, Susan Berman, Tristan Nixon, and Ambarish Joshi. 2014. Better data beats big data. In *Educational Data Mining 2014*.