37 Million Compilations: Investigating Novice Programming Mistakes in Large-Scale Student Data

Amjad Altadmri
School of Computing
University of Kent
Canterbury, Kent, UK
aa803@kent.ac.uk

Neil C. C. Brown
School of Computing
University of Kent
Canterbury, Kent, UK
nccb@kent.ac.uk

ABSTRACT

Previous investigations of student errors have typically focused on samples of hundreds of students at individual institutions. This work uses a year's worth of compilation events from over 250,000 students all over the world, taken from the large Blackbox data set. We analyze the frequency, time-to-fix, and spread of errors among users, showing how these factors inter-relate, in addition to their development over the course of the year. These results can inform the design of courses, textbooks and also tools to target the most frequent (or hardest to fix) errors.

Categories and Subject Descriptors

K.3.2 [**Computers And Education**]: Computer and Information Science Education

General Terms

Experimentation

Keywords

Programming Mistakes; Blackbox

1. INTRODUCTION

Knowledge about students' mistakes and the time taken to fix errors is useful for many reasons. For example, Sadler et al [10] suggest that understanding student misconceptions is important to educator efficacy. Knowing which mistakes novices are likely to make or finding challenging informs the writing of instructional materials, such as textbooks, and can help improve the design and impact of beginner's IDEs or other educatoinal programming tools.

Previous studies that have investigated student errors during [Java] programming have focused on cohorts of up to 600 students at a single institution [1, 4, 5, 7, 8, 13]. However, the recently launched Blackbox data collection project [3] affords an opportunity to observe the mistakes of a large number of students across many institutions – for example, in one year of data, the project collected error messages and Java code from around 265,000 users worldwide. A previous study by the authors utilized four months of data from Blackbox to study educators opinions against the frequency of mistakes [2]. The contribution in our proposed paper is to go further, and provide a more detailed investigation into characteristics of the mistakes, trying to answer the following research questions:

- What are the most frequent mistakes in a large-scale multi-institution data set?
- What are the most common errors, and common classes of errors?
- Which errors take the shortest or longest time to fix?
- How do these errors evolve during the academic terms and academic year?

SIGCSE'15, March 4–7, 2015, Kansas City, MO, USA.

http://dx.doi.org/10.1145/2676723.2677258.

2. RELATED WORK

The concept of monitoring student programming behavior and mistakes has a long history in computing education research. The series of workshops on Empirical Studies of Programming [11] in the 1980s had several papers making use of this technique for Pascal and other languages. More recently, there have been many such studies specifically focused on Java, which is also the topic of this study.

Many of these studies used compiler error messages to classify mistakes. Jadud [8] looked in detail at student mistakes in Java and how students went about solving them. Tabanao et al. [13] looked at the association between errors and student course performance. Denny et al. [4] looked at how long students take to solve different errors. Dy and Rodrigo [5] looked at improving the error messages given to students. Ahmadzadeh et al. [1] looked at student error frequencies and debugging behavior. Jackson et al. [7] identified the most frequent errors among their novice programming students. All six of these studies looked at cohorts of (up to 600) students from a single institution. These studies used compiler error messages to classify errors, while early results from McCall and Kölling [9] suggest that compiler error messages have an imperfect (many-to-many) mapping to student misconceptions.

Our study is novel in that it looks at student mistakes from a much larger number of students (over 250,000) from a large number of institutions[1], thus providing more robust data about error frequencies, and error commonality. The Blackbox work was originally presented with a brief list of the most frequent compiler error messages [3], but in this study we do not simply use compiler error messages to classify errors. Instead, we borrow error classifications from Hristova et al. [6], which are based on surveying educators to ask for the most common Java mistakes they saw among their students.

3. METHOD

3.1 Student Mistakes

We use the 18 mistakes from our previous study [2] as a basis for our analysis, which in turn were derived from Hristova et al's [6] twenty student mistakes (derived from interviewing educators). The eighteen mistakes, labeled A through R, are informally categorized as follows:

Misunderstanding (or forgetting) syntax:

- **A**: Confusing the assignment operator (`=`) with the comparison operator (`==`).
  For example: `if (a = b) ...`
- **C**: Unbalanced parentheses, curly or square brackets and quotation marks, or using these different symbols interchangeably.
  For example: `while (a == 0]`
- **D**: Confusing "short-circuit" evaluators (`&&` and `||`) with conventional logical operators (`&` and `|`).
  For example: `if ((a == 0) & (b == 0)) ...`
- **E**: Incorrect semi-colon after an if selection structure before the if statement or after the for or while repetition structure before the respective for or while loop.
  For example:
```
if (a == b);
    return 6;
```
- **F**: Wrong separators in for loops (using commas instead of semi-colons)
  For example: `for (int i = 0, i < 6, i++) ...`
- **G**: Inserting the condition of an if statement within curly brackets instead of parentheses.
  For example: `if {a == b} ...`
- **H**: Using keywords as method or variable names.
  For example: `int new;`
- **J**: Forgetting parentheses after a method call.
  For example: `myObject.toString;`
- **K**: Incorrect semicolon at the end of a method header.
  For example:
```
public void foo();
{
    ...
}
```

[1]We have no way of measuring the number of institutions in the Blackbox data, but simply: the 250,000 students must be split over at least several hundred institutions.

- **L**: Getting greater than or equal/less than or equal wrong, i.e. using `=>` or `=<` instead of `>=` and `<=`.
  For example: `if (a =< b) ...`
- **P**: Including the types of parameters when invoking a method.
  For example: `myObject.foo(int x, String s);`

Type errors:

- **I**: Invoking methods with wrong arguments (e.g. wrong types).
  For example: `list.get("abc")`
- **Q**: Incompatible types between method return and type of variable that the value is assigned to.
  For example: `int x = myObject.toString();`

Other semantic errors:

- **B**: Use of `==` instead of `.equals` to compare strings.
  For example: `if (a == "start") ...`
- **M**: Trying to invoke a non-static method as if it was static.
  For example: `MyClass.toString();`
- **N**: A method that has a non-void return type is called and its return value ignored/discarded.
  For example: `myObject.toString();`
- **O**: Control flow can reach end of non-void method without returning.
  For example:
```
public int foo(int x)
{
    if (x < 0)
        return 0;
    x += 1;
}
```
- **R**: Class claims to implement an interface, but does not implement all the required methods.
  For example: `class Y implements ActionListener { }`

Note that mistake *N* (ignoring the non-void result of a method) is not always an error (e.g. when you call a remove method that returns the item removed, you may not need to do anything with the return value).

3.2 Student data

Data about student mistakes was taken from the Blackbox data set [3], which collects Java code written by users of BlueJ, the Java beginners' IDE. We used data from the period 1st Sep. 2013 to 31st Aug. 2014 (inclusive), as representing a full year.

We had two methods of detecting mistakes. For four of the student mistakes, *I, M, O, R*, we were able to use the compiler error message directly from Blackbox's compilations to detect the mistake. However, this was not possible for the other errors, as some of them are logical errors that do not cause a compiler error or warning, while in other cases the error messages do not have a one-to-one mapping to our mistakes of interest. Thus for one of the other mistakes (C) we performed a post-lexing analysis (matching brackets) and for the final thirteen we used a customized permissive parser

to parse the source code and look for the errors. The source code of all the tools used is available for analysis or re-use [2].

We took each source file in the data set, and tracked the file over time. At each compilation we checked the source file for the eighteen mistakes. If the mistake was present, we then looked forward in time to find the next compilation where the mistake was no longer present (or until we had no further data for that source file). When the mistake was no longer found – which could have been because the mistake was corrected or because the offending code was removed or commented out – we counted this as one instance of the mistake. Further occurrences in the same source file were treated as further instances. We calculate the time in seconds between the first appearance of the mistake and the possible fix and cap it at 1000 seconds, for any mistake which takes longer than 1000 seconds to fix, or is never fixed. We also counted the number of distinct users who committed each mistake at least once.

4. RESULTS

Our detector is used to look for the number of instances of mistakes, as described in section 3.2. The data set featured 37,158,094 compilation events, of which 19,476,087 were successful and 17,682,007 were unsuccessful. Each compilation may include multiple source files – the total number of source files input to compilation (i.e. each source file may be counted multiple times) was 46,448,212, of which 24,264,603 were compiled successfully and 22,183,609 were not.

4.1 Mistakes Overview

The Frequencies, number of Distinct Users and Time-to-Fix results of each different mistake are shown in Table 1, along with our informal classification of the error message. It can be seen that although Mistake *C* is the most frequent, the type and semantic errors are generally more frequent than syntax errors. The ranking of frequency is similar to the ranking of the number of distinct users, as shown in Figure 1. This figure confirms that the number of users is related to the frequency of the mistakes. However, each mistake is not made the same number of times by each user that makes it. Figure 2 shows that some mistakes (*F, G, H, L*) are generally made only once by the user – a rare mistake, or one that they easily learn from. In contrast, mistakes such as *C*, *I* and *O* are made repeatedly by students – in the case of *C* (mismatched brackets), a syntax error, this is likely to be an easy accident, but in the case of *I* (wrong types in method call) and *O* (missing return statement) this is more likely to be a lack of understanding or difficulty with the concepts of types and control flow.

It is clear from Table 1 that, as expected, mistakes which don't result in a compiler error tend to take much longer to be fixed, such as Mistakes *N*, *B* and *D*. Mistake *E* gets fixed more quickly; although it does not cause a compile error, it has a more noticeable effect on the program's control flow.

To examine this relation more, Figure 3 shows a scatter plot for the mistake Frequencies against Time-to-Fix. If it was the case that students learned to quickly fix the most frequent errors, we would expect a trend along a diagonal line from top left to bottom right. This is not the case, showing that the time to fix an error is not noticeably related to how often a student will encounter it.

[2]See: `http://www.cs.kent.ac.uk/~nccb/blackbox/`

Mistake	Freq.	Distinct Users	Median Time to Fix	Error Type
C	793232	119407	17	Syntax
I	464075	66977	58	Type
O	342891	57129	37	Semantic
A	173938	45082	111	Syntax
N*	121663	27369	≥ 1000	Semantic
B*	121172	29627	≥ 1000	Semantic
M	86625	26527	48	Semantic
R	79462	17521	104	Semantic
P	52862	21904	24	Syntax
E*	49375	18350	401	Syntax
K	38001	19502	51	Syntax
D*	29605	8742	≥ 1000	Syntax
J	18955	10232	34	Syntax
Q	16996	7556	115	Type
L	4214	3317	12	Syntax
F	2719	2244	36	Syntax
H	1097	943	22	Syntax
G	118	112	26	Syntax

Table 1: Statistics for mistakes committed by Blackbox students between 1st Sep. 2013 to 31st Aug. 2014 (inclusive). The results are sorted according to frequency in descending order. Most mistakes will result in a compiler error message being shown to the user – those mistakes which do not cause a compiler error have an asterisk.

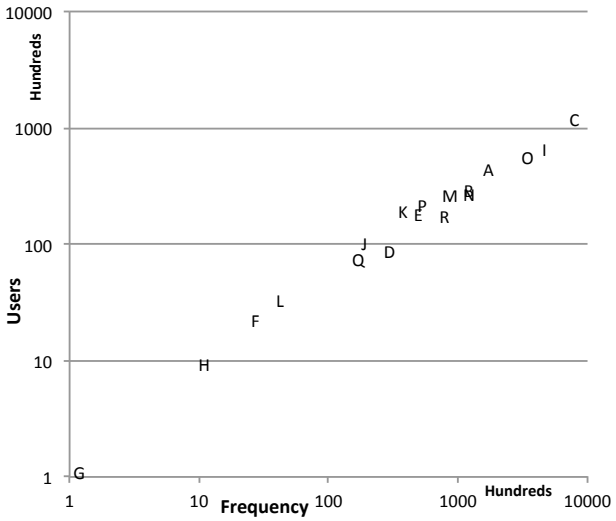

Figure 1: Log-log plot of Frequency vs Distinct users. Mistake *N* overlaps Mistake *B* towards the right.

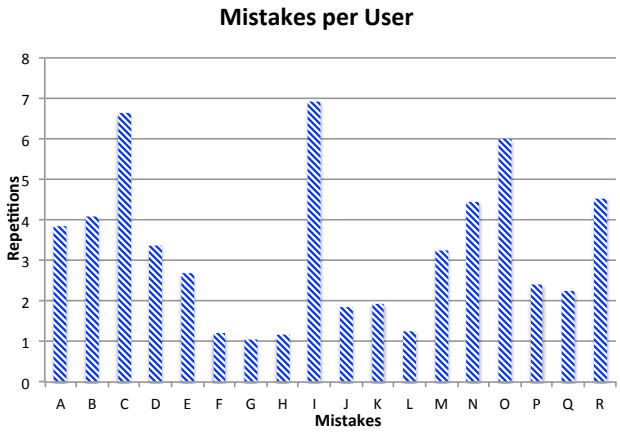

Figure 2: The average of number of times that each user who made a mistake, repeated it. For example, each user who made mistake *G* only made it approximately once, while each user who made mistake *B* made it four times on average.

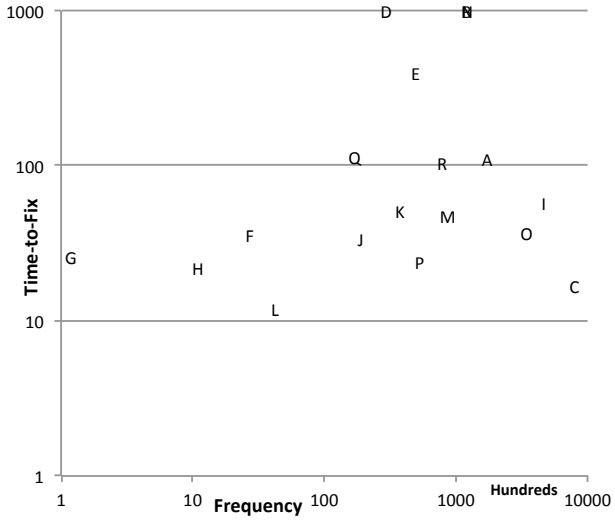

Figure 3: Log-log plot of Frequency vs Time-to-Fix. Mistake *N* overlaps Mistake *B* in the top right. Mistakes *N*, *B* and *D* have a median equal to the cap value of 1000, which means that more than half of the instances appeared took ≥ 1000 seconds to fix or were not fixed at all.

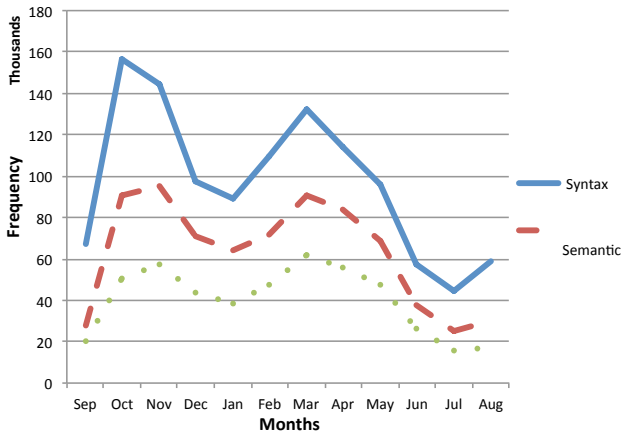

Figure 4: The development of frequency of mistakes categories over months (not normalised). The start of terms is very clear in all of the three categories.

4.2 Mistakes Over Time

To have a deeper look on these factors and to understand the effect of the learning process on these mistakes, we analysed the mistake factors over the course of the year, split into months. Figure 4 outlines the mistake frequencies for the three proposed categories over the course of 12 months, without normalization (i.e. raw figures, which are clearly affected by the number of compilations changing between months). The three curves show clear peaks in the number of errors (and underneath, the number of compilations) in October/November and March.

To allow a better comparison, the data is rendered again in Figure 5 after normalization: we divide the frequencies of the mistakes in each category in each month by the number of total compilations in that month. At the start of the data (and to a smaller degree at the turn of the calendar year), Syntax errors show a small peak at the beginning of the northern hemisphere academic year, but otherwise are fairly flat. In the contrast, the slight peaks for the Semantic and Type categories happen later. We propose that this is due to the order of topics in academic courses: students initially struggle with syntax, but as they master this, they begin to grapple instead with semantic and type errors.

As described in section 4.1, and shown in Table 1, Mistake *C* is clearly the most frequent mistake – but in general, Semantic mistakes were more frequent than Syntax mistakes. However, Mistake *C* is also one of the fastest mistakes to fix, and thus could be considered relatively unproblematic. Figure 6 shows the same data as Figure 5, but with mistake *C* removed. With this adjustment, Syntax errors are much less common, and show less of an early peak, suggesting that students learn, to some extent, to avoid making the *C* bracket mistakes.

This suggestion that students learn to avoid *C* is confirmed in figure 7, which shows the normalized number of users making each mistake (not the frequency). Mistake *C* shows a peak in Aug/Sep at the start of the northern hemisphere academic year, whereas many of the other errors have a fairly flat distribution across the 12 months. This suggests that users are not learning to completely avoid the non-*C* mistakes over the course of the year.

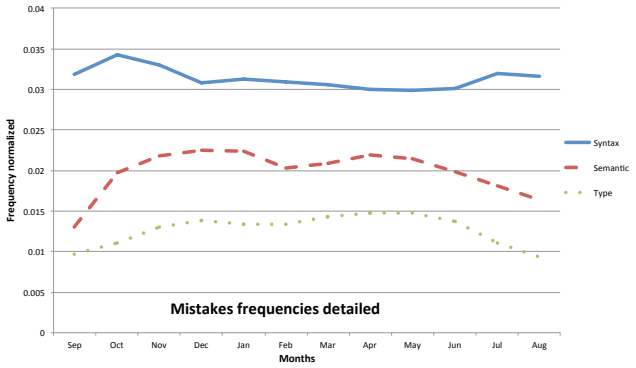

Figure 5: The development of normalized frequency of mistakes categories over months. While the Syntax errors tend to flatten after a while after each term starting, the Semantic and Type ones increase.

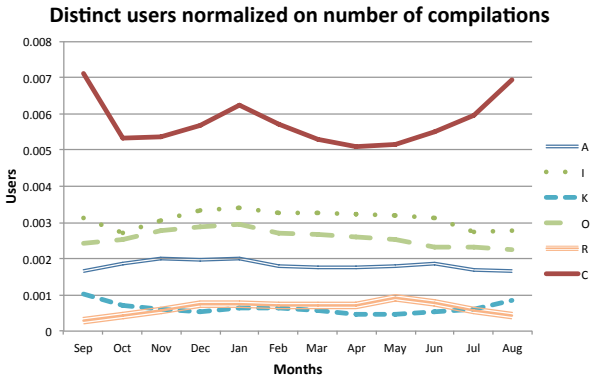

Figure 7: The development of normalized number of distinct users over months, for some selected mistakes. Omitted mistakes have almost the same shape as the ones presented, except C, and been removed for clarity only. The curve representing Mistake C is the most interesting one, showing that around quarter of students learn to avoid it towards the mid of the term.

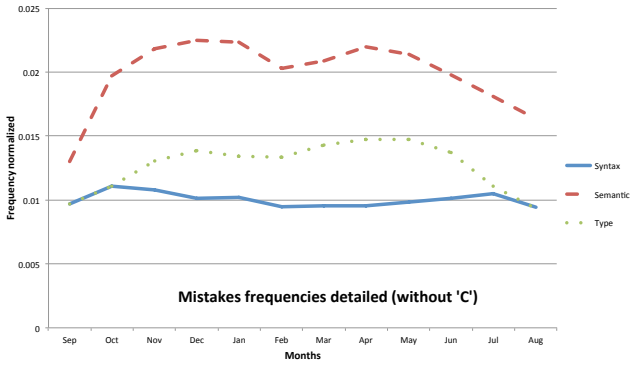

Figure 6: The development of normalized frequency of mistakes categories over months, after omitting Mistake C. The drop of the Syntax errors below the other two categories is very clear, maintaining its shape. This supports the importance of focusing more on the Semantic and Type errors.

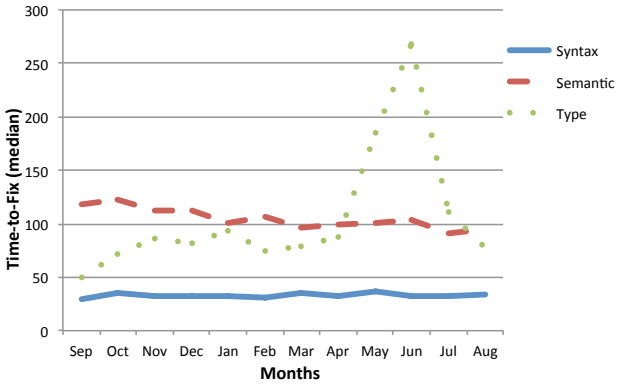

Figure 8: The development of the median of time-to-fix over months for the mistakes' categories. While the Syntax mistakes are almost quick to fix and consistent over time, the Semantic ones need more efforts and this decreases a little over time. The Type curve is interesting as it shows a sudden rise and drop towards the end of the academic year, possibly due to introducing advance types for students.

The time-to-fix statistics are shown in figure 8. Time-to-fix of Syntax mistakes is almost flat, showing that they are quick to fix, but students do not generally get faster at fixing them. In contrast, Semantic mistakes show a decrease of the time needed to fix over the course of the year, Type mistakes show a strange, unexpected shape. The only interpretation we can offer is that with the increase of topics introduced to students, moving from primitive types to generic types and inheritance, students tend to make more mistakes between these 'more difficult' types.

5. CONCLUSIONS

The large sample-size of data from the Blackbox project provides comprehensive information about the frequency, spread of errors among users and time-to-fix of different Java errors, shown in Table 1. Unlike many previous studies, the analysis of errors is decoupled from compiler error messages.

Mismatched brackets and quotations – a syntax error – are the most popular error category among novice programmers, but are also the quickest to be fixed. Other Syntax errors are less frequent than Semantic and Type errors. With the exception of mismatched brackets and quotations, Semantic errors tend to appear higher than the other types. Recent research has suggested that Java has syntax which is not much better than an arbitrary choice of syntax for learning [12], but perhaps the syntax of a language does not matter as much as semantics. It is also clear that some of the mistakes (especially *L*, *F*, *H* and *G*) are particularly low frequency, in contrast to Hristova et al.'s [6] original aim to find the top 20 errors.

Global starts and finishes of the northern hemisphere's academic terms are detectable, and they have an effect on the frequency of Mistakes and time needed to fix them. The frequency of different classes of mistake changes over time: Semantic mistakes tend to increase in frequency later on in courses, while Syntax mistakes decrease.

We should be careful not to over-interpret this result as it is greatly affected by which mistakes we included in the study, but our results suggest that semantic errors are a more serious challenge than syntax errors.

The mistakes in this study were automatically detected using a set of tools. If these tools were not sufficiently accurate in picking up mistakes, this could have affected our results, and are thus a threat to validity. To this end, the source code is freely available (see section 3.2) in case further investigation is warranted.

6. REFERENCES

[1] M. Ahmadzadeh, D. Elliman, and C. Higgins. An analysis of patterns of debugging among novice computer science students. In *Proceedings of the 10th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education*, ITiCSE '05, pages 84–88, New York, NY, USA, 2005. ACM.

[2] 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*, pages 43–50. ACM, 2014.

[3] 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.

[4] P. Denny, A. Luxton-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.

[5] T. Dy and M. M. Rodrigo. 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.

[6] M. Hristova, A. Misra, M. Rutter, and R. Mercuri. Identifying and correcting Java programming errors for introductory computer science students. In *Proceedings of the 34th SIGCSE Technical Symposium on Computer Science Education*, SIGCSE '03, pages 153–156, New York, NY, USA, 2003. ACM.

[7] J. Jackson, M. Cobb, and C. Carver. Identifying top Java errors for novice programmers. In *Frontiers in Education, 2005. FIE '05. Proceedings 35th Annual Conference*, Oct 2005.

[8] 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.

[9] D. McCall and M. Kölling. Meaningful categorisation of novice programmer errors. In *Frontiers In Education Conference*, pages 2589–2596, 2014.

[10] P. M. Sadler, G. Sonnert, H. P. Coyle, N. Cook-Smith, and J. L. Miller. The influence of teachers' knowledge on student learning in middle school physical science classrooms. *American Educational Research Journal*, 50(5):1020–1049, 2013.

[11] E. Soloway and S. Iyengar, editors. *Empirical Studies of Programmers: Papers Presented at the First Workshop on Empirical Studies of Programmers*. Intellect Books, 1986.

[12] A. Stefik and S. Siebert. An empirical investigation into programming language syntax. *Trans. Comput. Educ.*, 13(4):19:1–19:40, Nov. 2013.

[13] E. S. Tabanao, M. M. T. Rodrigo, 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.