All Syntax Errors Are Not Equal

Paul Denny, Andrew Luxton-Reilly, Ewan Tempero
Department of Computer Science
The University of Auckland
Auckland, New Zealand
{paul,andrew,ewan}@cs.auckland.ac.nz

ABSTRACT

Identifying and correcting syntax errors is a challenge all novice programmers confront. As educators, the more we understand about the nature of these errors and how students respond to them, the more effective our teaching can be. It is well known that just a few types of errors are far more frequently encountered by students learning to program than most. In this paper, we examine how long students spend resolving the most common syntax errors, and discover that certain types of errors are not solved any more quickly by the higher ability students. Moreover, we note that these errors consume a large amount of student time, suggesting that targeted teaching interventions may yield a significant payoff in terms of increasing student productivity.

Categories and Subject Descriptors

K.3.2 [**Computers and Education**]: Computer and Information Science Education—*computer science education*

General Terms

Design, Human Factors

Keywords

Syntax errors, Java, CodeWrite, drill and practice, assessment

1. INTRODUCTION

Syntax errors are frequently encountered by the novice programmer. As educators, the more we understand about the nature of these errors and how students respond to them, the more effective our teaching can be [12]. In particular, if there are certain errors that students spend much of their time struggling with, targeted support may be effective.

Previous research examining students working on typical Java programming assignments, involving multiple class and method definitions, has shown that a few types of syntax errors occur much more frequently than most [4, 6]. While the time elapsed between successive compilations is generally short [5], it is not known if students spend more time solving *certain kinds* of syntax errors than others.

In this paper, we examine the syntax errors encountered by students writing short fragments of Java code when implementing single methods. Despite differing from previous work in the length of the code being written, we confirm earlier results that a small number of errors are highly represented. We also show that students spend a large proportion of their time fixing problems associated with these most common errors, and that the highest performing students take as long to correct certain errors as any other students.

2. RELATED WORK

Understanding syntax has long been a source of difficulty for programming students [11]. Many novices find it frustrating to struggle with syntax errors [7], and given that programming is a subject in which the cumulative effects of previous learning are critical to progression [10], this has the potential to hinder long-term success.

A varied body of research concerning how novices encounter and respond to syntax errors exists. A number of tools have been developed and evaluated to help students fix syntax errors in their programs. SyntaxTrain [9] parses a student's source code and, if it detects a syntax error, displays a syntax diagram illustrating the required syntax. Another example in this category is the Expresso tool [3], which operates as a pre-compiler, producing detailed messages from a set of pre-defined common syntax errors.

Other research has explored, in controlled conditions, how students approach syntax error correction. Kummerfeld et al. [7] deliberately inserted errors into a set of 10 programs and then observed how a small number of students, of varying programming experience, went about correcting them. Participants were provided with a web-based guide cataloguing a number of common errors that they could use as a reference. Among their conclusions they report that more experienced programmers adopt clear strategies for detecting and fixing errors, and that for all students syntax error correction can be very time consuming.

More closely aligned with the work we present here, several studies examine and categorize the error messages actually generated by students when programming. Reports of this nature tend to use technology to automatically capture students' compilations. Jackson et al. [4] utilize their web-enabled Gauntlet IDE, which logs all student errors while a session is active, sending the full collection of errors to a remote server once the student exits the IDE. They tabulate the most common errors, and report that based on teaching experience, faculty successfully predicted 5 of the 10 most common errors.

Jadud [5] used a modified version of the popular BlueJ IDE that collected relevant information every time a student compiled their program on campus. The collected data included a snapshot of the complete program and the result of each compilation. The most common errors were reported, as well as the average time taken between successive compilation events. The two most common errors, "Missing ;" and "Cannot resolve identifier/symbol", match the result of Jackson et al. although their order is reversed. There is also some agreement amongst the less frequent errors, however a comparison is complicated by the fact that the two studies define different error categories.

Previous work has not attempted to measure the time that students spend solving *particular kinds* of syntax errors. Our expectations were that more capable students would identify and solve syntax errors more quickly than their weaker counterparts. In addition to our intuitive feelings about this, it is supported by previous research in which the differences between the kinds of tasks achievable by students in different performance quartiles has been examined. In the domain of code comprehension, Lister et al. [8] applied the SOLO taxonomy to describe how novice programmers manifest their understanding of code. Their data show that the more capable students are far more likely to successfully integrate parts of a problem into a coherent structure. They propose that this difference may account for the particular difficulty students in the lower quartiles of a class experience when writing code.

3. METHODOLOGY

In this paper, we analyse the syntax error messages encountered by students when writing short fragments of Java code. The data were collected using CodeWrite [1], a web-based drill and practice tool with which students tackle exercises requiring the completion of a single method body. As a consequence, submissions are typically short, and students focus on just the syntax and logic for the one method being implemented without concern for larger program design issues. Figure 1 illustrates the length of all submissions in our data set containing at least one syntax error. The average over all such submissions is 6.5 lines of code.

When a student chooses an exercise to answer, they are shown the method signature (the return type, method name, and names and types of the parameters), a textual description of the purpose of the method, and the expected output for one example input. Students type and edit their code directly in the browser, and submit when they are ready to receive feedback. Their submitted code is compiled on the server, and if syntactically correct, is executed against a set of test cases. If the code contains syntax errors, the corresponding error messages are displayed.

We report here on data collected from the Semester 1 (March-June) offering of the COMPSCI 101 (CS1 in Java) course taught at the University of Auckland in 2011. Of the 478 students enrolled, 430 sat the final exam (a 10% withdrawal rate is typical). Each student was given approximately three weeks to correctly answer 30 exercises on CodeWrite for which they could earn a total of 2% credit towards their final grade in the course. This period of participation spanned parts of Weeks 6 and 7 (of the 12 week teaching semester) and included a two-week mid-semester break. At the point of the course that the CodeWrite activity was introduced, students had already completed their first programming assignment and attended 10 hours of supervised lab sessions. Importantly, while the second half of the course covers object-oriented design, for the period in which the data was collected for this study Java was only used for teaching basic programming concepts such as primitive and array variables, control structures, basic functions from the Java API, and methods and parameter passing.

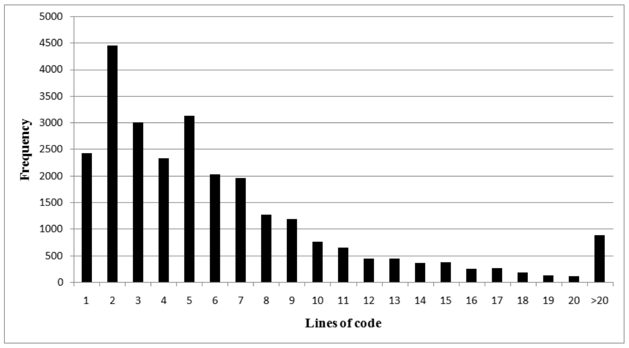

Figure 1: Length of all submissions containing at least one syntax error

In the following sections, we use the term *submission* to represent a single fragment of code submitted to one exercise by one student, and the term *exercise attempt* to represent the full set of submissions made by a single student to a particular exercise. Each submission can be classified into one of the three types listed in Table 1.

Table 1: Submission types

Type	Explanation
"P"	the submission compiles and passes all of the exercise test cases
"F"	the submission compiles but does not pass all of the exercise test cases
"X"	the submission does not compile

Following each submission, a student receives immediate feedback, which in the case of an "X" submission would include the error messages from the compiler. Note, there may be several error messages generated for each submission of type "X". The failing test cases are displayed for a submission of type "F". A timestamp records when each submission is received by the CodeWrite server.

As an example, Figure 2 shows an exercise attempt consisting of four submissions made by one student when solving an exercise to calculate the volume of a cube given the side length. Upon making their first submission, the student receives a "Cannot resolve identifier" type error for not declaring the variable `volumeOfACube`. For their next submission, they replace the expression calculating the volume with a call to `Math.pow()`. In this case, they receive the same error message as before, as the compiler still cannot resolve the identifier `volumeOfACube`. They then declare `volumeOfACube` to be of type `int`, which exposes the "type mismatch" error introduced by the use of `Math.pow()`. The student suc-

```
Submission type: X (9 Mar 2011, 11:36:59 PM)
Error: volumeOfACube cannot be resolved

public int volumeOfACube(int side) {
        volumeOfACube=side*side*side;
        return volumeOfACube;
}
-------------------
Submission type: X (9 Mar 2011, 11:37:36 PM)
Error: volumeOfACube cannot be resolved

public int volumeOfACube(int side) {
        volumeOfACube=Math.pow(side,3);
        return volumeOfACube;
}
-------------------
Submission type: X (9 Mar 2011, 11:39:59 PM)
Error: Type mismatch: cannot convert from double to int

public int volumeOfACube(int side) {
        int volumeOfACube;
        volumeOfACube=Math.pow(side,3);
        return volumeOfACube;
}
-------------------
Submission type: P (9 Mar 2011, 11:40:29 PM)

public int volumeOfACube(int side) {
        int volumeOfACube;
        volumeOfACube=side * side * side;
        return volumeOfACube;
}
```

Figure 2: Example of an exercise attempt for one student

cessfully corrects this mistake by reverting to the expression they used previously.

A student who completed the requirements for the activity as described previously would therefore have made at least 30 exercise attempts, where each one includes at least one submission of type "P". Figure 3 illustrates the data we have collected from this course for a single student that is relevant to the analysis presented in this paper.

3.1 Research questions

We know that students of all levels of ability frequently write code that does not compile, and that weaker students are often unable to solve their syntax problems [2]. Our interest in this paper is to investigate the kinds of syntax errors that students are most commonly making, and measure the time they spend solving these errors. This also provides us with an opportunity to replicate the results of earlier studies in this area, but in a context where students are writing short fragments of code. We outline our three research questions in this section.

> RQ1: Which syntax errors do students most commonly encounter, and how does this compare with previous work?

Our examination of the common types of syntax errors is based upon the error messages generated by the compiler rather than on inspection of the code. To tabulate the error frequencies we analysed all submissions of type "X" and, for each, processed every error message generated by the compiler. Error messages were aggregated and duplicates

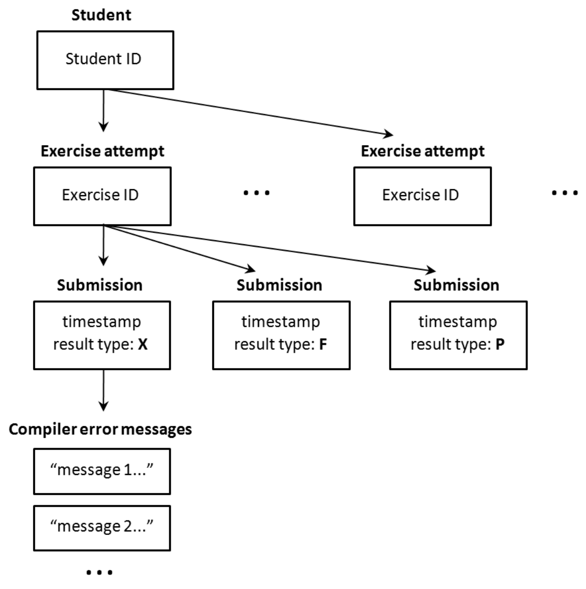

Figure 3: Data collected for each student

removed, so that for any given submission, a particular *type* of syntax error was either present or not. For example, if two variable identifiers, `i` and `j`, were used in one submission without first being declared, then error messages "`i` cannot be resolved" and "`j` cannot be resolved" would both be generated by the compiler. Likewise, if the variable identifier `i` appeared in multiple places in a submission without first being declared, the compiler would generate the error message "`i` cannot be resolved" for each occurrence. In either case, we classified the multiple error messages under the single, more general, type "Cannot resolve identifier".

When defining syntax error categories, we attempted to align them with previously published results where possible, however there has not been agreement on this in the past. For example, in [5] the category "bracket expected" is reported, whereas four separate categories "`{` expected", "`(` expected", "`}` expected" and "`)` expected" are reported in [4].

> RQ2: How long do the different kinds of syntax errors take students to solve, and does this vary by level of ability?

It is difficult to measure precisely, through a purely automated analysis, how long it takes a given student to successfully fix a given error that occurs during an exercise attempt. Our analysis relies on the assumption that if a submission contains a particular type of error, then the compiler will generate a corresponding error message. This is often, but not always, the case. For example, certain types of errors in a submission may prevent the compiler from discovering and listing all errors.

Given this assumption, the first "X" submission of an exercise attempt that generates a particular type of error represents the point at which the error was introduced into the code. Of all subsequent submissions, the first one that does not generate the same type of error represents the point at which the error was corrected. A timestamp records when every submission is made. Therefore, the *time difference* between the submission that first generates the error, and

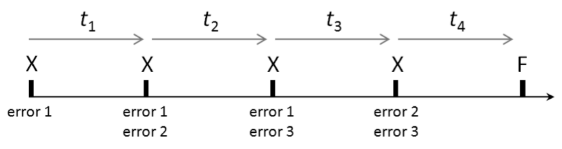

Figure 4: An example exercise attempt consisting of five submissions

the next submission in which the error is not generated is an approximate measure of how long the student spent solving that error. Of course, for any given submission there may be multiple errors generated (about 30% of "X" submissions generated more than a single error message), and a student may focus on fixing just one of these errors for every submission they make. Nevertheless, the period of time described above is a measure of the length of time that a particular error existed in the student's code prior to being corrected.

For example, the diagram in Figure 4 illustrates the timeline of an exercise attempt consisting of five submissions (X, X, X, X, F). The first submission of type "X" generates a single kind of error, whereas the next three submissions each generate two kinds of errors. The arrows, t_i, shown in the diagram represent the time differences between consecutive pairs of submissions. In this particular example, the time taken to correct "error 1" is $t_1 + t_2 + t_3$, and the time taken to correct "error 3" is $t_3 + t_4$. There are two separate occurrences of "error 2", the first of which took t_2 and the second of which took t_4 to correct. In this case, the average time taken to correct "error 2" is therefore $(t_2 + t_4)/2$.

To investigate this research question, for every kind of syntax error, we calculate the average time taken by each student across all of their submissions to correct that error. We only consider students who encountered the error at least once when calculating this average.

As student participation is unsupervised, we must be aware of the fact that an individual time measurement may be inflated if a student is interrupted or leaves a task and then returns to it at a later time. A simple heuristic for handling such cases is to ignore time periods greater than a particular threshold. A suitable threshold value would be one that captures a high percentage of the data yet represents a reasonable maximum period of time for a student to be working on preparing a submission.

To determine possible threshold values, we examined the elapsed times between all consecutive pairs of submissions for all exercises. A histogram of this data is shown in Figure 5. The majority of elapsed times are short, nearly half (45%) being less than 30 seconds, and the frequency of the periods reduces rapidly with their length. Roughly 8% of the periods are longer than 380 seconds.

We repeated our analysis 8 times using threshold values between 120 and 540 seconds inclusive (at 60 second intervals). For simplicity in presenting the data in this paper, unless otherwise specified, a threshold value of 300 seconds has been used.

We were interested in how the time spent solving errors varied by student ability. Students were split into quartiles based on their performance in the course overall. The final course mark was based on assignment, laboratory, midterm test and final exam marks. Our analysis only considers students who participated with CodeWrite, and because students in the lower quartiles were less likely to participate, there are fewer students in these quartiles. Table 2 summarizes the number of students who sat the exam, the number who participated with the CodeWrite activity, the total number of exercises attempted, the total number of submissions, and the number of submissions of type "X" for students in each quartile (Q1 is the top quartile).

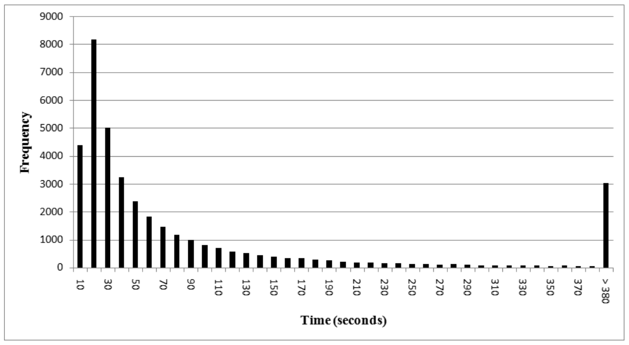

Figure 5: Elapsed times between consecutive submission pairs

Table 2: Student participation data by quartile

	All	Q1	Q2	Q3	Q4
Sat exam	430	108	107	107	108
Participated	385	106	105	97	77
Exercises	15481	4923	4281	3814	2463
Submissions	53888	18022	15917	12282	7667
... of type "X"	26387	7853	8138	6305	4091

RQ3: How much student time, in total, was spent on the most common syntax errors?

Focusing our teaching efforts on the types of syntax problems for which students are spending a disproportionate amount of time may help to assist learning and increase student productivity. To explore this idea, we determined the total amount of time that all students spent on each error, and the average amount of time spent per student on each error.

4. RESULTS

4.1 RQ1: Most common errors

Of the 53888 submissions made by all students, 26387 contained at least one syntax error (i.e. were "X" submissions). The most common type of error, "Cannot resolve identifier", appeared in 6344 of these submissions. Table 3 lists the ten most common error messages, the total number of submissions in which they each occurred, and the percentage of all "X" submissions that this represents.

These results agree moderately with previous work. Our three most common error messages appear 2nd, 7th and 1st in Jadud's table [5] and 1st, 7th and 2nd in Jackson's table [4], respectively. We note that "Type mismatch" errors occured much more frequently in our study than in previous work. One possible explanation for this is the nature of the exercises students work on. As every exercise involves implementing a method body, students must always ensure

Table 3: Most common syntax errors (all students)

Error type	Total	%
Cannot resolve identifier	6344	24.0%
Type mismatch	4847	18.4%
Missing ;	3436	13.0%
Token should be deleted	2712	10.3%
Method not returning correct type	1743	6.6%
Missing }	1450	5.5%
Missing)	1213	4.6%
Missing {	512	1.9%
Using .length as a field	433	1.6%
Insert "AssignmentOperator"	339	1.3%

that the type of the return expression matches the return type of the method. We noticed many errors as a result of this kind of mismatch.

We also note that the Sun/Oracle JDK compiler was used in both of the previous studies. CodeWrite uses the batch compiler that is part of the Java Development Tools (JDT) Core component of the Eclipse development environment. The error messages produced by this compiler, which evolved from IBM's VisualAge environment, differ from those produced by the Sun/Oracle JDK. For example, consider the following expression with unmatched parentheses:

```
double result = Math.round(value * 100)) / 100.0;
```

The Sun/Oracle JDK produces the following error message:

```
';' expected
```

whereas the JDT batch compiler (used by CodeWrite) would generate:

```
Syntax error on token ")", delete this token
```

A full comparison of error messages is beyond the scope of this paper, however, as in this example, we often found the messages produced by the JDT batch compiler more informative than those produced by the Sun/Oracle JDK.

Despite these differences, although in our study students were practicing with short fragments of code, they still frequently encountered the same kinds of errors as in previous research where students were developing complete programs.

4.2 RQ2: Time to solve errors by quartile

We measured the time a student took to solve a particular type of syntax error as the time that elapsed while the corresponding error message was generated for consecutive submissions of type "X". Figure 6 shows the time students in different quartiles spent, on average, solving errors of the three most common varieties.

Both "Cannot resolve identifier" and "Type mismatch" errors took longer for students to solve than "Missing ;" errors. As expected, more capable students tended to spend less time solving "Missing ;" errors than weaker students (students in the top quartile, Q1, solved these errors more than twice as quickly as students in the bottom quartile). Interestingly, this did not appear to be the case for the other two errors. To investigate the significance of any differences between quartiles we ran a Kruskal-Wallis test for independent samples (this non-parametric method was selected because the samples did not have similar variances and ANOVA is not robust to violations of this assumption when the sample sizes are unequal).

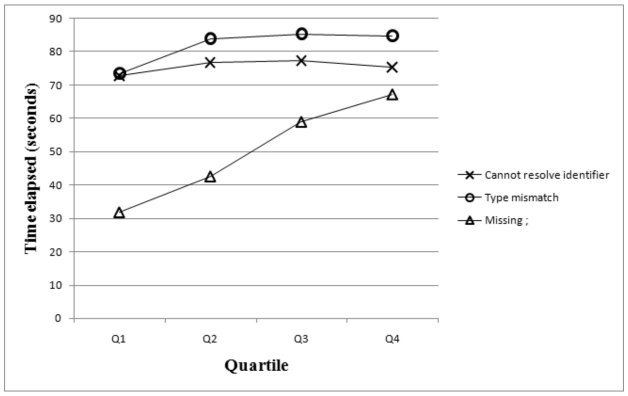

Figure 6: Average time spent correcting submissions (with a single error message) by quartile

The differences between quartiles for the "Missing ;" error were highly significant ($p \ll 0.001$), whereas the time spent solving "Cannot resolve identifier" errors did not vary by quartile (p = 0.534). These were very robust results, and held across all threshold values we investigated.

For "Type mismatch" errors, students in the top quartile solved them marginally more quickly than other students however these differences were not significant (p = 0.073). This held when the analysis was performed using threshold values less than 300 seconds, however for threshold values greater than 300 seconds the differences were significant at the 0.05 level ($0.027 < p < 0.050$).

In summary, although more capable students spend far less time solving certain kinds of errors than other students, somewhat surprisingly this is not the case for all kinds of errors.

4.3 RQ3: Cumulative time solving errors

Throughout the practice activity for which we have collected data, not all students encountered every kind of syntax error. The total amount of time a student spent tackling a particular type of error was dependent both on how frequently the error occurred and how long it took to solve.

Table 4 summarises, for each of the most common kinds of errors, the number of students that encountered the error at least once, the cumulative total time all students spent solving errors of that kind (in hours) and the average amount of time per student spent solving errors of that kind (in minutes). Of the 385 students who participated, 356 encounterd at least one "Cannot resolve identifier" error. This error stands out, along with the next most common type of error, "Type mismatch", as consuming a particularly large amount of student time – twice that of any other error.

5. DISCUSSION

Our interest was in understanding where beginners spend most of their time due to failed compilation when implementing short methods. From earlier work in this area, we expected to see a few kinds of errors making up the majority, and we confirmed previous results in this respect. We speculated that as these errors were so common, the amount of *time* spent on them may have been proportionally less, how-

Table 4: **Number of students, total time by all and average time per student spent on each error**

Error type	*n*	Total (hrs)	Avg. (mins)
Cannot resolve identifier	356	70.7	11.9
Type mismatch	341	57.1	10.0
Missing ;	347	26.8	4.6
Token should be deleted	316	26.7	5.1
Method not returning correct type	268	22.6	5.1
Missing }	234	14.3	3.7
Missing)	251	11.5	2.8
Missing {	110	4.7	2.6
Using .length as a field	118	4.9	2.5
Insert "AssignmentOperator"	90	3.6	2.4

ever the two most common errors also dominated students' time. Finally, we predicted more capable students would solve all types of errors in less time than the weaker students in class. Although this was sometimes the case, quite unexpectedly we found that all students spent a similar amount of time solving the most common errors no matter what quartile they were in.

That all students struggle equally with these errors, and spend so much time with them, suggests there is some fundamental issue that needs to be addressed. It also suggests that providing specific support for these errors may be particularly useful.

Given the surprising nature of our result, we must consider other possible explanations (or, threats to validity) for it. One possible factor is that when a student makes a submission of type "X" that contains multiple errors, we do not know their strategy for correcting those errors. Some students may attempt to fix all errors before making their next submission, while others may attempt to fix what they perceive is the simplest error first. To investigate this, we repeated our analysis considering only submissions of type "X" for which a single error message was generated. In other words, we ignored the 30% of "X" submissions containing more than one kind of error that may contribute to this threat. This analysis did not affect our conclusions.

It should also be noted that students are free to choose which exercises they answer. It is possible that students with lower capability tend to choose easier exercises than those with higher capability. Although this may be true, there is no obvious reason why the "Cannot resolve identifier" and "Type mismatch" errors are any harder (or easier) to solve due to the difficulty of the exercise. However this might be true of other errors – for example it may be harder to determine where the missing "}" should go in more complex code than in simple code.

6. CONCLUSIONS AND FUTURE WORK

In this paper, we investigated the types of errors most frequently encountered by students practicing writing short fragments of Java code using the CodeWrite practice tool. We discovered, confirming previous work in different contexts, that students encounter a small number of errors much more frequently than others.

Our expectations were that the most capable students in class would be able to resolve errors more quickly than less able students, and this result was confirmed for one of the top three most frequent errors, the classically well-known "Missing ;". However, quite unexpectedly, we found that for the two most common errors, the time taken to correct the corresponding mistakes did not vary by quartile. This result indicates that specific teaching support around the causes of these errors may be particularly effective.

In future work, we will develop interventions to provide teaching support for these common mistakes, and measure their success against the baseline data collected in this study.

The data reported here was collected over a fairly short period of time that included a mid-semester break and coincided with just one week of class time. Additional future directions will examine the way in which student responses to particular types of errors change over time as their learning progresses and their experience grows.

7. REFERENCES

[1] P. Denny, A. Luxton-Reilly, E. Tempero, and J. Hendrickx. CodeWrite: Supporting student-driven practice of Java. In *Proceedings of SIGCSE '11*, pages 471–476, 2011.

[2] P. Denny, A. Luxton-Reilly, E. Tempero, and J. Hendrickx. Understanding the syntax barrier for novices. In *Proceedings of ITiCSE '11*, pages 208–212, 2011.

[3] M. Hristova, A. Misra, M. Rutter, and R. Mercuri. Identifying and correcting Java programming errors for introductory computer science students. *SIGCSE Bull.*, 35:153–156, January 2003.

[4] J. Jackson, M. Cobb, and C. Carver. Identifying top Java errors for novice programmers. In *Proceedings of ASEE/IEEE Frontiers in Education Conference*, FIE '05, pages T4C24–T4C27, 2005.

[5] M. C. Jadud. A first look at novice compilation behaviour using BlueJ. *Computer Science Education*, 15:1–25, 2005.

[6] M. C. Jadud. Methods and tools for exploring novice compilation behaviour. In *Proceedings of ICER '06*, pages 73–84, 2006.

[7] S. K. Kummerfeld and J. Kay. The neglected battle fields of syntax errors. In *Proceedings of ACE '03*, pages 105–111, 2003.

[8] R. Lister, B. Simon, E. Thompson, J. L. Whalley, and C. Prasad. Not seeing the forest for the trees: novice programmers and the solo taxonomy. *SIGCSE Bull.*, 38:118–122, June 2006.

[9] A. L. A. Moth, J. Villadsen, and M. Ben-Ari. SyntaxTrain: relieving the pain of learning syntax. In *Proceedings of ITiCSE '11*, pages 387–387, 2011.

[10] A. Robins. Learning edge momentum: A new account of outcomes in CS1. *Computer Science Education*, 20:37–71, 2010.

[11] D. Sleeman, R. T. Putnam, J. A. Baxter, and L. Kuspa. *An introductory Pascal class: A case study of students' errors*, pages 207–235. Lawrence Erlbaum Associates, Hillsdale, NJ, USA, 1988.

[12] J. C. Spohrer and E. Soloway. Novice mistakes: are the folk wisdoms correct? *Commun. ACM*, 29:624–632, July 1986.