A New Metric to Quantify Repeated Compiler Errors for Novice Programmers

Brett A. Becker
School of Computer Science
University College Dublin
Belfield, Dublin 4, Ireland
brett.becker@ucd.ie

ABSTRACT

Encountering the same compiler error repeatedly, particularly several times consecutively, has been cited as a strong indicator that a student is struggling with important programming concepts. Despite this, there are relatively few studies which investigate repeated errors in isolation or in much depth. There are also few data-driven metrics for measuring programming performance, and fewer for measuring repeated errors. This paper makes two contributions. First we introduce a new metric to quantify repeated errors, the repeated error density (RED). We compare this to Jadud's Error Quotient (EQ), the most studied metric, and show that RED has advantages over EQ including being less context dependent, and being useful for short sessions. This allows us to answer two questions posited by Jadud in 2006 that have until now been unanswered. Second, we compare the EQ and RED scores using data from an empirical control/intervention group study involving an editor which enhances compiler error messages. This intervention group has been previously shown to have a reduced overall number of student errors, number of errors per student, and number of repeated student errors per compiler error message. In this research we find a reduction in EQ, providing further evidence that error message enhancement has positive effects. In addition we find a significant reduction in RED providing evidence that this metric is valid.

General Terms

Algorithms; Measurement; Human Factors; Languages

Keywords

Java; debugging; errors; syntax errors; error messages; compiler errors; repeated errors; consecutive errors; error quotient; EQ; feedback; novice; programming; CS1

1. INTRODUCTION

Many studies on novice programmer behavior analyze errors along with other facets of the programming experience, with far fewer focusing extensively or exclusively on errors, particularly repeated errors. There are also few data-driven metrics to measure programming performance [15], and fewer for measuring repeated errors. This is despite the fact that the number and frequency of repeated errors has been cited as the best indicator of how well a programming student is progressing [11]. Most studies that focus on repeated errors investigate using metrics to predict student achievement, normally by correlating metrics with assessment marks. However, the quality and usefulness of these correlations have varied considerably. In addition, the metric normally used, Jadud's Error Quotient (EQ) [10, 11], has been recently shown to vary across groups, environments, and contexts, with further validation called for before the EQ can be used as a proxy for traditional measures of performance [12].

In that light, this paper introduces a new metric for measuring repeated errors, the *Repeated Error Density* (RED). We also demonstrate that this metric is less context dependent, useful for short sequences, and is significantly reduced by an editor that has previously been shown to result in significantly fewer compiler errors (including repeated errors). Being less context dependent is seen as important since being able to generalize on study results is a fundamental requirement for putting them into effective use [15]. The authors are not aware of any other study empirically comparing the number/frequency of repeated errors between control and intervention groups.

This paper is laid out as follows. In Section 2 we review related work involving repeated error metrics. In Section 3 we present a new metric, the *Repeated Error Density* (RED), and demonstrate that it has properties which enable it to answer some questions posited by Jadud in [11], which the Error Quotient cannot. In Section 4 we apply the RED to real-world data and compare it to the EQ. We show that RED is reduced in an empirical control/intervention study, providing evidence that this metric is valid. Reductions in EQ and RED also support the results of [3] and [4], providing further evidence that enhancing error messages is effective. We also discuss some threats to validity. In Section 5 we present concluding remarks and future work.

2. RELATED WORK

It has been shown that the majority of students spend the majority of their time on a minority of errors and that students spend more time solving certain kinds of syntax errors than others [9, 17]. In addition, repeating errors is common, and has been observed and studied for at least the last four

ITiCSE '16, July 09-13, 2016, Arequipa, Peru
© 2016 ACM. ISBN 978-1-4503-4231-5/16/07…$15.00
DOI: http://dx.doi.org/10.1145/2899415.2899463

decades, and across languages such as: assembly [8], C [15, 19], Java [3, 11, 15], LISP [14], Pascal [13] and Python [15]. In [10] it was found that when a student encounters certain high frequency errors in Java, there were often double-digit chances that the next error was the same error. Several reasons for students' frequently repeating errors have been postulated:

- Some students may be meticulous in their programming, and checking (possibly more often than necessary) that their code compiles, or that the state of play has not changed since their last thought [10].
- Some students may be sloppy in their interaction with the compiler. In some cases, interaction with the program source may cause the most recent error to disappear, requiring a recompile for a reminder [10].
- Some students may not trust the error message reported by the compiler, and are recompiling in a naive hope that the output will change [10].
- There is evidence that experienced programmers do not make as many repeated errors as novices [14], and that beginning students make more repeated errors than more advanced students [8].
- Students may be struggling with the nature of particular errors, a fundamental programming misconception [13], the actual compiler error message being presented to them, and/or the syntax of the language itself [11].

As early as 1984 repeated errors began to be noticed by those interested in novice programmer behavior. PROUST [13] analyzed Pascal programs written by novices, diagnosed non-syntactic bugs, and explained them to the programmer. When a bug was found, it was not initially known if it was for instance a typographical accident, or the result of a deep-rooted misconception. The authors of [13] noted that distinguishing between these two explanations required looking for the same bug being repeated; if a bug occurred just once it was most likely an accident, if it occurred repeatedly the case for a misconception being the cause was strengthened.

The authors of [14] studied a group of students programming in LISP. These students knew LISP 'reasonably well', in that their errors could be considered 'slips' as defined in [5]. These students were unlikely to repeat the same error within a single program or across different programs.

In [2], the authors analyzed over 37 million Java compilation events from the Blackbox data set [7]. In particular they analyzed the frequency of 18 errors also explored in [6]. They found that the number of times each of these errors was repeated by the same student (not necessarily consecutively), in the same file, varied between 1 and 7. On average it could be expected that a given student can expect to repeat an error that was previously committed in the same file a further 3.3 times.

The authors of [8] developed a model of students studying assembly language which included five stages: novice, advanced beginner, competent, proficient, and expert. Each stage was characterized by a number of debugging abilities and habits. The novice and advanced beginner stages were partly characterized by repeating the same types of error throughout a program *frequently* and *occasionally* respectively. Repeating errors was not listed as an ability/habit of the three more advanced stages.

In [19] a framework was proposed to support the education of novice programmers. Among many features, the framework sought to aid educators in helping students who kept repeating the same error(s) as these students in particular needed to be provided a long time for student support. In this framework when a student repeatedly made the same mistake, the identity of the student and the error were relayed to the instructor. This feature was seen to work well in a trial with students programming in C.

In [10], Jadud found that many Java errors were frequently repeated (consecutively) by novices. In particular he found that 21% of *; expected*, 17% of *unknown variable*, 20% of *bracketing* 30% of *illegal start of expression*, and 14% of *unknown class* errors were followed by the exact same error, representing consecutive repeats.

In [11], Jadud reported that the number of repeated errors was an indication of struggling students, and often the best indicator for how well (or poorly) a student was progressing. To measure this behavior, Jadud proposed the *Error Quotient* (EQ), a quantification for novice compilation behavior that provides an indication for how much a student is struggling with the language while programming [10, 11]. Specifically, the EQ is a value that is deterministically generated from observable (loggable) data. The EQ is normalized to between 0 and 1 inclusive, and averaged over a programming session. A low EQ indicates that a student is dealing with errors they encounter efficiently. A high EQ indicates that a student is struggling, making changes that do not rectify the error causing a particular compiler error message. Importantly, it was found that students with a low EQ generated relatively few compiler error messages while those with a high EQ generated many compiler error messages that were *repeated*, often in the same location and over many successive compilations. It was found that the EQ correlated significantly with traditional indicators of academic success such as coursework assessments and end-of-year exams. The EQ is determined by Algorithm *EQ* [11] which utilizes Figure 1.

Algorithm ***EQ***

```
1. Collate Create consecutive pairs from all compilation
   events in a session, e.g. $(e_1, e_2), (e_2, e_3), ..., (e_{n-1}, e_n)$.
2. Calculate Score each pair according to Figure 1.
3. Normalize Divide the score assigned to each pair by
   11 (the maximum value possible for each pair).
4. Average Sum the scores and divide by $n$, the number
   of pairs.
```

The EQ is very dependent on the amount of repeated errors. In fact if every compilation in a session ends with the generation of the exact same compiler error, the EQ for that session would be 1. If no errors are generated or all errors are fixed on the first attempt, the EQ would be 0. If there are repeated errors of differing types, the EQ would be somewhere between 0 and 1.

The penalties assigned in Figure 1 are not arbitrary. In fact the original EQ algorithm scored several more parameters including error and edit locality [11]. In an attempt to find the best parameters, Jadud sought to discover those which differentiated between students across the whole population by choosing the parameter set that had the greatest range for each student between the minimum and maximum

EQ values while minimizing the standard deviation of EQ values for each student.

In [10], Jadud found a distinct correlation between a student's error quotient and the grades they received on assignments. Despite the correlation being significant it was described as being of low quality ($R^2 = 0.11$). The correlation between error quotient and final exam grade was found to be more significant, but also of low quality ($R^2 = 0.25$).

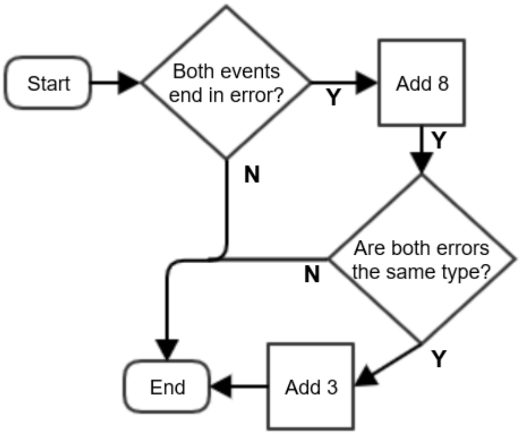

Figure 1: The error quotient event pairing algorithm. Values are then normalized and averaged according to Algorithm *EQ*.

In [11], Jadud noticed that the average error quotient dropped throughout the course of a year-long study, indicating that this population learned to deal with syntax errors more effectively with time. In trying to establish if the error quotient could be used to predict student assessment marks, a significant correlation between the error quotient and marks received on an exam given at the end of the first semester of observation was found, although of poor quality ($R^2 = 0.22$). Unfortunately there was no significant correlation found between the error quotient and an exam given at the end of the year.

Rodrigo et al. [16] asked if they could identify at-risk novice programmers and predict student achievement using the EQ metric. They found a moderate but significant correlation with mid-term exam scores. The authors of [18] also found a correlation between EQ scores and achievement on midterm exams, but their models could not accurately predict at-risk students.

In [15] the authors explored both versions of the EQ with data sets from four CS1 contexts involving different working practices, student backgrounds and languages: C, Python and Java. Using linear models the authors also explored a range of parameter values. These models relate average EQ to a measure of student performance, either a summative assessment or course mark. Overall they found that different contexts produced very different models. In fact the Python and Java data sets were best served by almost opposing models. The authors also found that the Java data set did not benefit from the additional EQ parameters, similar to conclusions reached by Jadud [10]. In addition their C data set did not yield a significant model. However the explanatory power of both Python data sets was improved. Nonetheless even data drawn from two contexts with the same language (Python) resulted in extremely different models.

The authors concluded that context specific variables such as programming tools (IDEs, etc.), language, and type of work students are asked to perform while gathering data all greatly affect the EQ and its predictive power. Of course there are a multitude of other factors that may also have effects such as demographics, physical environment, etc. These results suggest that data-driven metrics such as the EQ need to be tuned for their specific contexts.

Watson et al. [20] also found a significant correlation between the EQ and the scores of students in their data ($r = 0.44$). They proposed an extended algorithm called Watwin which takes into account the time a student spends in overcoming a particular error. Like the EQ, the Watwin score is between 0 and 1 inclusive, with a score of 0 indicating that no errors were made over a session. A score of 1 indicates that every compilation ended in an error, and that the student spent longer than their peers between successive compilations. The closer the score is to 0, the stronger the student is deemed to be. With this improved algorithm the correlation improved to ($r = 0.51$).

The authors of [1] explored how the EQ and Watwin metrics perform for detecting high- and low-performing students. However they observed that the EQ and Watwin had relatively poor performance on the data at their disposal.

Finally, several avenues for putting the error quotient to further use are proposed in [11] including being used by tutors, students, peer groups, in interactive and passive tutorials, and as a real-time tool and formative evaluation mechanism. We have similar visions for the Repeated Error Density proposed in Section 3.

In this section we have reviewed that repeating errors is common among novices, the frequency with which errors are repeated is an indicator of how well a student is progressing, and that students require help with repeating errors. We also saw that the correlation between the EQ and traditional performance metrics is varied, and sometimes tenuous, and finally that the EQ itself is in need of further study and validation. Recent work [12] has gone a long way toward such, but specifically the impact of context on the EQ still leaves unanswered questions.

3. THE REPEATED ERROR DENSITY

In [11], Jadud asked "If one student fails to correct an 'illegal start of expression' error over the course of three compilations, and another over 10, is one student [about] three times worse than the other? What if the other student deals with a non-stop string of errors, and the repetition of this particular error is just one of many?" In these cases, both students would have an EQ equal to 1. Although the actual difference between these students is most likely forever debatable, is there a metric for repeated errors that does show a difference from which some meaning can be taken?

To answer this we must acknowledge that it is possible for repeated errors to manifest themselves in different ways. Let us start with the simplest case. Take student A, who logs two consecutive errors of type x, resulting in one repeated error r. Is it fair to say that student B who logs three consecutive errors of type x resulting in two repeated errors $2r$ is struggling with that error more, by a factor of two? What about student C who also logs $2r$, but not all in one string s (say across $2s$, by logging two consecutive errors of type x followed by some successful compiles, followed by two more consecutive errors of type x)? Assuming that no other errors are made, and all other variables among the three students are equal, the EQ will be equal to 1 in each case. Table 1 summarizes these three scenarios.

Table 1: Three students repeating error x in different scenarios. Student A repeats x once ($1r$), while students B and C repeat x twice ($2r$). Student B logs $2r$ in one string s, while Student C does so in $2s$. In all cases, EQ = 1.

student	**sequence S**	x	r	s	**EQ**
A	... x x ...	2	1	1	1
B	... x x x ...	3	2	1	1
C	... x x ... x x ...	4	2	2	1

In trying to answer Jadud's questions stated at the beginning of this section, Table 1 shows that the EQ is not of any help. What we are after is a metric that could tell us what seems intuitive - that student C is struggling twice as much as student A with error x (in as much as student C's sequence is two of Student A's strung together). Student B seems to be somewhere in the middle - having struggled with error x for longer than student A, but once it was overcome it did not recur as it did for student C. Then again, this student fought error x for a longer consecutive string of events than Student B. Does this indicate that this student had more trouble with error x overall?

The search for a new metric to give insight into the above went down many avenues. One involved looking at the amount of repeated error strings a student encounters. Another looked at the length of these strings. Ultimately a metric emerged which incorporates both the number of strings and their lengths. We call this metric the *Repeated Error Density* (RED). This entails summing a submetric calculated on each repeated error string encountered in a sequence of compilation events. This submetric is $r_i^2/|s_i|$ where $|s_i|$ is the length of string s_i containing r_i repeated errors. This can be expressed completely in terms of r_i as $r_i^2/(r_i+1)$, since the length of a string s_i, containing r_i consecutive errors is always equal to $r_i + 1$.

The value of RED for a given sequence S of n repeated error strings is the sum of $r_i^2/(r_i+1)$ for each string s_i in S, given by Equation 1, where r_i is the number of repeated errors in string s_i. Note that we define a repeated error r as a pair of events where each event results in the same error; in other words consecutively committing the same error. Table 2 shows the values of RED for all sequence combinations for which a total of 0-4 repeated errors occur.

$$RED = \sum_{i=1}^{n} \frac{r_i^2}{r_i+1} \tag{1}$$

The RED has properties which can help us answer Jadud's questions. The first question was "If one student fails to correct an 'illegal start of expression' error over the course of three compilations, and another over 10, is one student three times worse than the other?" Let us rephrase this to: If one student repeats error x once, and another repeats error x twice, is the latter student struggling two times more than the former? Sequence 3 in Table 2 is the only way that error x can be repeated once. A single occurrence of error x (sequence 1), or two occurrences separated by other activity (sequence 2), do not constitute any (consecutive) repeats.

Table 2: Values of RED for all sequence combinations for 0-4 repeated errors r. A, B, C correspond to students in Table 1.

no.	**sequence S**	r	**RED**
1	x	0	0
2	... x ... x ...		0
3 (A)	... x x ...	1	0.5
4 (C)	... x x ... x x ...	2	1
5 (B)	... x x x ...		$1.\overline{3}$
6	... x x ... x x ... x x ...	3	1.5
7	... x x x ... x x ...		$1.8\overline{3}$
8	... x x x x ...		2.25
9	... x x ... x x ... x x ... x x ...	4	2
10	... x x ... x x ... x x x ...		$2.\overline{3}$
11	... x x x ... x x x ...		$2.\overline{6}$
12	... x x ... x x x x ...		2.75
13	... x x x x x ...		3.2

There are however two ways that error x can be repeated twice consecutively, shown by sequences 4 and 5. Sequence 4 is the 'natural' choice for representing twice the struggle of sequence 3, as sequence 4 is exactly two sequences equal to sequence 3 strung together. Table 2 shows that RED = 1 for sequence 4, and 0.5 for sequence 3, a ratio of 2:1. This property applies to any two similarly related sequences; for example sequences 11 (RED = $2.\overline{6}$), and 5 (RED = $1.\overline{3}$), again a ratio of 2:1. Thus, according to the RED metric we can answer Jadud's first question affirmatively.

Jadud's second question was "What if the other student deals with a non-stop string of errors, and the repetition of this particular error is just one of many?" Table 2 shows that sequence 5, containing one 'long' string of two repeated errors has a RED = $1.\overline{3}$. Sequence 4 which also has two repeated errors but over two shorter strings has a RED = 1. In this case the RED metric assigns a penalty of 1/3 for encountering two repeated errors in one longer string, compared to the same amount of repeated errors in a greater number of shorter strings. Going back to our students in Table 1, we can say that according to the RED metric, student B is struggling more with error x than student C. Both have two repeated errors, but student B struggled with error x for a longer consecutive string of events than student C. This is reflected in a higher RED score.

Following is a non-exhaustive list of properties of the Repeated Error Density metric. RED...

1. specifically and exclusively measures repeated errors.
2. does not depend on parameter values, which must be matched to context and subject to debate, unlike the EQ.
3. accounts for the amount of repeated errors in a sequence.
4. accounts for the amount of repeated error strings a sequence.
5. accounts for the lengths of the repeated error strings in a sequence.
6. has a value of 0 for sequences with 0 repeated errors.
7. is additive: e.g. sequence 6 is sequence 3 + sequence 4, and RED for sequence 6 is equal to RED of sequence 3 + RED of sequence 4 - see Table 2.

8. is proportional: values of related sequences are proportional (e.g. sequences 3, 4; 3, 6; 5, 11 in Table 2).
9. assigns higher penalties for higher repeated error density (e.g. sequences 6, 7, 8 in Table 2).
10. $\in \mathbb{R}_{\geq 0}$, unlike EQ and Watwin which are both $\in [0, 1]$.

4. APPLICATION TO STUDENT DATA

To compare and contrast the Error Quotient and the Repeated Error Density, and to see how the RED behaves on real data, we applied each to a data set obtained with the methodology described in [4]. This data consists of 29,019 error events from two cohorts of programming novices (approximately 100 students each) who had compiler data logged for four weeks while using a custom Java editor. The control group experienced the normal JDK compiler errors while the intervention group experienced enhanced errors provided by the editor based on the JDK error and the student code. This editor was found, with the same data set, to significantly reduce the overall number of student errors, the number of errors per student, and the amount of repeated student errors per compiler error message [4].

Figure 2 shows a histogram of EQ values for the control and intervention groups. Shapiro-Wilk tests confirmed normal distributions for EQ of both groups. A Student's t-test (two-tail) showed that the mean EQ was reduced by approximately 15% for the intervention group ($M = 0.33$, $SD = 0.16$) compared to the control group ($M = 0.39$, $SD = 0.18$); $t(210) = 2.70$, $p = 0.008$.

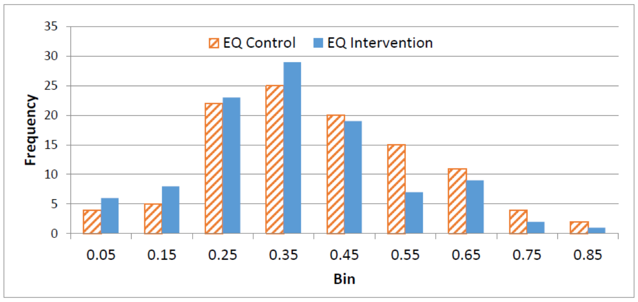

Figure 2: Histogram of EQ values for control and intervention groups.

Figure 3 shows a histogram of RED values for each group. A Shapiro-Wilk test confirmed that the control group was normal. However the intervention group was skewed right (skewness 0.76). A Mann-Whitney U test (two-tail) showed a 31% reduction in median RED for the intervention group ($Mdn = 34.0$) compared to the control group ($Mdn = 49.4$); $U = 3,638$, $p < 0.001$.

Of course RED is just one of many possible metrics that could be developed to measure repeated errors. However as RED takes into account not only the amount of repeated errors, but the number of strings in which they occur, and the lengths of these strings (the density), it allows us to answer Jadud's questions presented in Section 3.

As Jadud and Dorn pointed out in [12], the EQ has little predictive power with a small number of data points, which Table 1 confirms. On the other hand, the RED is very useful with a small number of data points, as Table 2 shows. However the unbounded nature of the RED brings some questions when a large number of data points are involved, as Figure 3 shows. Despite a significant reduction in RED, the presence of outliers, particularly toward the right, brings some questions. For instance it is evident that these outlying students did struggle with repeated errors, but how much of their RED scores is reflected simply by a large number of compilations? In other words, the more a student programs, the more errors that student is likely to make. This will, as repeated errors are encountered, increase the overall RED score. As mentioned in [12] and extensively explored in [15], the definition of a session can have a large impact on EQ, and on parameter choice. This could be the reason that our observed EQ scores (particularly for the control group) are slightly higher than in some other studies. We defined a single session to be the time a student is working on a single file. We observed that our students were not only working on lab assignments, but simply practicing their programming. Unlike some previous studies they were given no starter code. This could explain higher numbers of errors, including repeated errors compared to other studies. It is possible the the RED metric is more suitable for shorter sessions, while the EQ is more suitable for longer sessions.

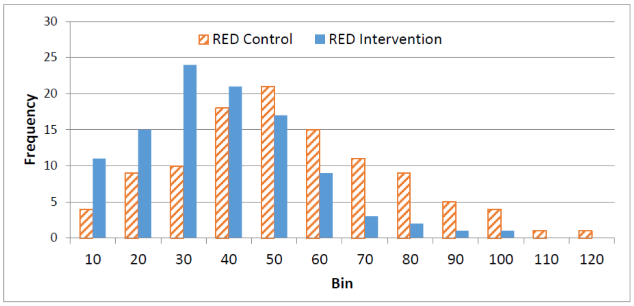

Figure 3: Histogram of RED values for control and intervention groups.

We see it as a positive that the RED does not depend on parameter values unlike the EQ. Jadud spent considerable time exploring the EQ parameter space in [11], and in [15] it was found that useful parameter values are likely to be highly context dependent. This makes the RED metric attractive as it can be applied in any context without depending on parameter value selection.

We also see it as positive that the EQ and RED were reduced for the intervention group, adding to the evidence in [3] and [4] that enhancing compiler error messages can be effective.

Finally, the threats to validity discussed at length in [3] and [4] would also apply here. We feel that this work gives merit to the RED metric being worth further exploration, particularly where programming sessions are shorter, and when starter code is provided. It also shows theoretical promise, backed up with our initial experimental work here, as a less context dependent repeated error metric than the EQ.

5. CONCLUSIONS AND FUTURE WORK

Repeated errors, particularly those committed consecutively, have been shown to be a strong indicator of students struggling with programming. Several studies have investigated repeated errors, most of which utilize Jadud's Error Quotient (EQ). However the EQ has been shown to be context dependent and can have a large parameter space. Additionally, the EQ has limited use for short sessions with few errors. Recently, the predictive power of the EQ has been shown to be inconsistent and sometimes tenuous.

We have introduced an alternative metric to measure repeated errors which we call the Repeated Error Density (RED). This metric shows promise as a repeated error metric that is less context dependent due to being based only on the number of consecutively repeated errors and the number of repeated error strings in a compilation event sequence. Being less context dependent is seen as important since being able to generalize on study results is a fundamental requirement for putting them into effective use [15]. We have demonstrated that the RED is effective on short sequences and that it can answer two questions (which the EQ cannot) that Jadud asked when first proposing the EQ.

We applied the RED to existing control/intervention data and found that the RED is reduced by an editor that provides enhanced error messages, consistent with previous findings demonstrating that this editor also reduced overall errors, overall number of student errors, number of errors per student, and number of repeated student errors per compiler error message. This provides evidence that this metric is valid.

Finally, a reduction in the EQ and RED for the intervention group adds to the evidence in [3] and [4] that enhancing compiler error messages can be effective.

Future work involves investigating and validating the RED metric in greater detail, particularly in different contexts, including those involving short programming sessions and where starter/skeleton code is provided. We also plan on investigating if there is a correlation between RED and success in assessments.

6. REFERENCES

[1] A. Ahadi, R. Lister, H. Haapala, and A. Vihavainen. Exploring machine learning methods to automatically identify students in need of assistance. In *Proceedings of the Eleventh Annual International Conference on International Computing Education Research*, pages 121–130. ACM, 2015.

[2] A. Altadmri and N. 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*, pages 522–527. ACM, 2015.

[3] B. A. Becker. *An exploration of the effects of enhanced compiler error messages for computer programming novices.* Master's thesis, Dublin Institute of Technology, 2015.

[4] B. A. Becker. An effective approach to enhancing compiler error messages. In *Proceedings of the 47th ACM Technical Symposium on Computer Science Education*, pages 126–131. ACM, 2016.

[5] J. S. Brown and K. VanLehn. Repair theory: A generative theory of bugs in procedural skills. *Cognitive science*, 4(4):379–426, 1980.

[6] N. 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.

[7] 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*, pages 223–228. ACM, 2014.

[8] R. Chmiel and M. C. Loui. Debugging: from novice to expert. *ACM SIGCSE Bulletin*, 36(1):17–21, 2004.

[9] P. Denny, A. Luxton-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*, pages 471–476. ACM, 2011.

[10] M. C. Jadud. A first look at novice compilation behaviour using bluej. *Computer Science Education*, 15(1):25–40, 2005.

[11] M. C. Jadud. *An exploration of novice compilation behaviour in BlueJ.* PhD thesis, University of Kent, 2006.

[12] M. C. Jadud and B. Dorn. Aggregate compilation behavior: Findings and implications from 27,698 users. In *Proceedings of the eleventh annual International Conference on International Computing Education Research*, pages 131–139. ACM, 2015.

[13] W. L. Johnson. *Intention-based diagnosis of novice programming errors.* Morgan Kaufmann, 1986.

[14] I. R. Katz and J. R. Anderson. Debugging: An analysis of bug-location strategies. *Human-Computer Interaction*, 3(4):351–399, 1987.

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

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

[17] J. C. Spohrer and E. Soloway. Novice mistakes: Are the folk wisdoms correct? *Communications of the ACM*, 29(7):624–632, 1986.

[18] E. S. Tabanao, M. M. 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*, pages 85–92. ACM, 2011.

[19] H. Tamada, A. Ogino, and H. Ueda. A framework for programming process measurement and compiling error interpretation for novice programmers. In *Software Measurement, 2011 Joint Conference of the 21st Int'l Workshop on and 6th International Conference on Software Process and Product Measurement (IWSM-MENSURA)*, pages 233–238. IEEE, 2011.

[20] C. Watson, F. W. Li, and J. L. Godwin. 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*, pages 319–323. IEEE, 2013.