PSU–Journal of Engineering, Technology, and Computing Sciences (JETCS)
(Vol. No. 2, Issue 2 pp. 8-15, December 2020)

The Relationship of Compilation Behavior Metrics and Student Performance in Introductory Programming Course

Christine Lourrine S. Tablatin[1], Ma. Mercedes T. Rodrigo[2]
Pangasinan State University[1]
Ateneo de Manila University[2]
ctablatin@psu.edu.ph[1], mrodrigo@ateneo.edu[2]

Abstract – *How students cope with syntax errors is a possible indicator of students' programming skills or levels of understanding. Researchers have therefore taken an interest in studying student online protocols, hence the development of online protocol analysis metrics such as Error Quotient (EQ), Watwin Algorithm, and Repeated Error Density (RED). These metrics aim to quantify how well students cope with syntax errors which could be used to predict students' performance. Previous studies were conducted comparing the predictive power of these metrics, however, the limited number and inconsistencies of the results require further exploration to come up with consistent and reliable results. We compare three data-driven metrics to determine which among them predicts students' midterm exam scores. The findings showed that RED could significantly predict students' midterm scores and accounted for 12.2% of the explained variability in midterm exam scores. The result of the comparison of the three metrics showed that EQ is the better predictor among them since it accounted for 20.2% of the explained variability in midterm exam scores.*

Keywords – *Compilation behaviors, data-driven metrics, Error Quotient, Watwin, Repeated Error Density.*

INTRODUCTION

Programming is a difficult skill for students to learn. A literature review conducted by Watson and Li [14] reports that about 32% of the students worldwide fail their CS1 or introductory programming course. Computer science educators and researchers over the past years attempted to determine and understand where student difficulties lie. They found that students have difficulty understanding programming concepts [1, 9], tracing, reading, and understanding pieces of code [5, 10].

How students cope with syntax errors is a possible indicator of students' skills or levels of understanding. Researchers have therefore taken an interest in studying student online protocols, defined as the set of all program submissions to the compiler. The online protocols are typically gathered through an instrumented interactive development environment. Each time a student compiles a program, the program source code as well as the event type (success or fail), timestamp, error message reported, line number, and other information is saved to a database that can later be analyzed [13, 11].

Three examples of online protocol analysis metrics are the Error Quotient (EQ)[7], Watwin Algorithm [16], and Repeated Error Density [3].

Jadud explores students' compilation behavior by dynamically capturing "snapshots" of their programs using the BlueJ IDE which were taken every time they compile their Java program. Analyzing these snapshots provided information about how students write their programs. EQ is a metric that quantifies how students fix their syntax errors [2].

A compilation event is represented as one record which consists of the event type, error message, the location of the error in the file, and the source code [13]. Some studies include more compilation event parameters which are needed in their analysis. To calculate the EQ [7], we create compilation pairings by taking consecutive pairs from the set of compilation events of each student. We then score each pair of compilation events according to the algorithm presented in Figure 1. The scores are normalized by dividing the sum of the total scores assigned to each pair of compilation events by 9 which is the maximum score possible for each pair. Finally, compute the sum of the scores and then divide it by the number of pairs. The average which ranges from 0 to 1 is taken as the EQ score for the programming session.

Since the introduction of EQ as a measure of compilation behavior of novice programmers in 2006, researchers have become more interested in the collection of log data on students' programming processes while they are working on their laboratory exercises or course assignments. To test the predictive power of EQ, Jadud [8] conducted a study correlating students' EQ and their performance on traditional assignments and final exams using code snapshots of the students' program. The findings of the study revealed that the quality of the relationship between EQ and performance in assignments is poor because only 11% of the variance in the performance can be explained by EQ, though the fit is significant. The relationship between EQ and the final exam is more significant but still considered to be poor since EQ can only account for 25% of the variance of students' final exam scores.

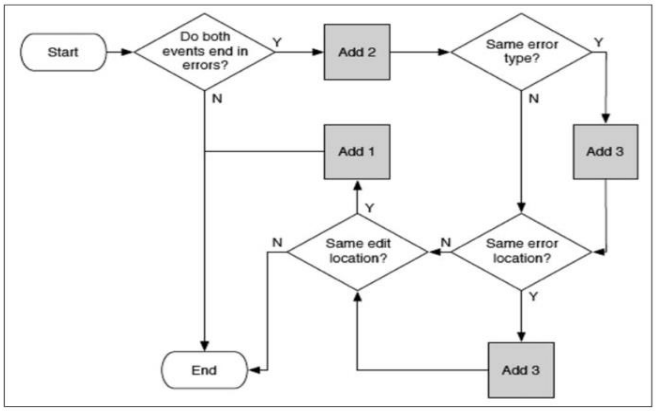

Figure 1. EQ Algorithm Flowchart

A similar study was conducted by Rodrigo, et al. [12] which correlated the mean EQ score with the students' midterm score. This was considered as one of the first successful use of EQ as a predictor of student performance since they arrived at a moderate but significant correlation value R= -0.54 (p<0.001). Further exploration was conducted in the study of Tabanao, et al. [13] to find out which among the errors encountered, the average time between compilation and EQ are predictive of students' midterm scores. It was found that compared to the linear regression models using errors encountered and time between compilations, the model with EQ as a factor is considered to be a highly significant predictor of the midterm score. The model could account for a moderate amount of the variance in performance which is 29.7% compared to 13.87% for the errors encountered and 6.51% for the average time between compilations. These studies provided evidence of the predictive power of EQ using students' programming behavior.

The findings on the studies regarding EQ proved that it is a weak predictor of students' performance. Watson, et al. [16], saw methodological weaknesses in the approach used by the EQ algorithm. It was noted that EQ assumes that students work on a single source file while in reality, students can work on multiple files at the same time. In between compilations of different files would result in inaccurate compilation pairings since the construction of pairings in EQ used the natural order of the compilation events during a programming session. Further, EQ did not consider the amount of time spent by a student in resolving their errors. The attempt to address the limitations of EQ led to the development of the Watwin algorithm. The limitation on the construction of compilation pairings was addressed by constructing the pairings on a per-file basis, ordered by timestamps. In addition, the algorithm incorporates a scoring approach that considers the amount of time spent by students to resolve a specific type of error compared to the resolve time of their peers.

Similar to EQ, Watwin also requires the construction of consecutive pairings for each file a student is working on and estimates the amount of time a student has spent in resolving an error. A more detailed approach in preparing the set of compilation pairings can be found in [16]. The scoring algorithm assigned penalty based on their resolve times which can be one standard deviation above the mean or below the mean. A higher penalty is assigned to longer resolve time which indicates that students have fixed an error much slower than their peers, while a lower penalty is assigned to those who resolve their errors quickly. The components included and the assigned penalties were based on the result of previous studies that used regression models and cross-validation and were not a product of random guesswork. This makes the algorithm applicable to independent datasets.

To evaluate the predictive power of the developed algorithm, the Watwin score was correlated to the overall coursework mark. The findings revealed that the Watwin score could significantly predict performance, which could explain 30% of the variance in overall coursework marks. The same dataset used in the analysis of the predictive power of Watwin was applied to EQ to compare their predictive power. The result was consistent with other studies that EQ is a weak predictor of performance having explanatory power between 15%-20% of the variance in performance. Further comparison of EQ and Watwin was conducted in [15] which is consistent with the findings of [16] wherein both are predictive of students' performance and that Watwin is a stronger predictor than EQ. However, recent exploration of the metrics in the study of Carter, et al. [4] presented a surprising result that was drastically different from the results of previous studies comparing EQ and Watwin. The result found out that EQ can only explain 3% of the variance in students' final grade compared to 18% on the study of [15] while Watwin could only account for 12% compared to 36%. The results of these studies provided evidence that Watwin is a stronger predictor compared to EQ.

A third algorithm for online protocol analysis quantifies repeated errors by looking at the number of error strings a student encounters as well as the length of these strings. This metric is less context-dependent and is suitable for short programming sessions. RED has properties that could answer the following questions of Jadud [7]: (1) "If one student fails to correct an 'illegal start of expression error throughout three compilations, and another over 10, is one student [about] three times worse than the other?", and (2) "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?". RED could answer the first question by providing a means to quantify how much a student struggles with a particular error by looking at the length of repeated error strings of that particular error compared to their peers. The second question could also be answered by the metric since it assigns a higher penalty to one longer repeated error strings compared to two shorter repeated error strings.

The algorithm to compute the RED score is simple. The metric involves summing up a sub-metric calculated on each repeated error string encountered in a compilation event sequence. This sub metric 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 ri as $r_i^2/(r_i + 1)$, since the length of a string s_i, containing 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 the equation:

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

where r_i is the number of repeated errors in string s_i.

We have to take note that a repeated error r is defined as a pair of events where each event results in the same error, which means committing the same error consecutively.

RED has been compared to EQ to check how the metric behaves on real data which was collected from the compiler data recorded for four weeks while students used a customized Java editor. The data consists of 29,019 error events from two groups of novice programmers with approximately 100 students each. The results showed that RED is more suitable for shorter sessions, while EQ is more suitable for longer sessions. It was also pointed out, that RED could be a valid metric when compiler error messages are enhanced. However, no studies have been conducted yet to determine if RED correlates with student performance which provides an opportunity for further research.

These metrics share a single purpose: to quantify how well students cope with syntax errors. Their precise methods differ (to be discussed in the methods section). There are limited studies, though, comparing the performance of each of these metrics against the other. The limited number of studies that were conducted to prove the predictive power of EQ and Watwin as well as the inconsistency of research findings of previous researches calls for further exploration of these metrics. While evidence that Watwin outperformed EQ in predicting students' performance in the contexts in which they were used, the introduction of RED as a new metric in quantifying students' compilation behavior provided opportunities to evaluate RED as a performance predictor and see if it can outperform Watwin and EQ.

OBJECTIVES OF THE STUDY

The goal of this study is to determine how accurately these metrics predict students' performance in an introductory programming course. This study aims to answer the following research question:

1. Which among the data-driven metrics used to quantify students' programming behavior accurately predict their performance?

MATERIALS AND METHODS

The data used in this study were gathered during the First Semester of the School Year 2013-2014 from students enrolled in CS21A - Introduction to Computing 1, which is the first programming course of Computer Science and Management Information System students in one of the universities in Quezon City. There were 80 students who have given their consent to participate in the study and agreed to have their programming process recorded. The participants were informed about the study on the first day of their class and were asked to sign a consent letter indicating that they were willing to be part of it. They were not obliged to join the study since participation is voluntary.

The laboratory exercises were performed by the students using the machines in the computing laboratories which have the same operating system, Java standard development kit, and BlueJ. These exercises were recorded using the data gathering tool developed by [8] which was implemented as an extension to the BlueJ development environment. The laboratory exercises were given during the participants' laboratory schedule and were designed to be finished for one hour. The compilation events of each participant were recorded and stored in a database where each record represents one compilation event per student.

The data that were gathered were then pre-processed for analysis. The fields that were extracted for data analysis were the unique session identifier, the error message, the line number where the error is located, the source code, the time of compilation, the filename, and the event type of every file compilation made by the students. The records that were extracted were stored in an Excel file which was later used in generating summaries. Further, the records were examined to determine which data to be included or excluded in the analysis. Online protocol records that were not related to the exercise were excluded. The records with less than seven compilation events and the amount of time spent which is less than 10 minutes in solving the exercise which indicates that students gave up on the laboratory exercise were also excluded. These records were not included since they do not constitute a representative sample of the students' compilation behavior.

These data were then analyzed using EQ, Watwin, and RED to quantify student compilation behaviors which are discussed in the preceding sections.

Scoring of Compilation Events of a Programming Session

To illustrate how the scores of each compilation event in a programming session are computed using the data-driven metrics, we would refer to Table 1 which can be assumed as a complete programming session. The table consists of compile number, timestamp, event type, error message, error location, and filename. All of these elements are needed to compute the Watwin Score. To compute the EQ score, we use the compile number, event type, error message, and error location. In contrast, the RED score calculation only needs the compile number, event type, and error message.

Calculation of EQ Score

EQ is a metric used to quantify how well the students cope with syntax errors. To compute for the EQ score, we follow the algorithm presented in Figure 1 wherein a corresponding penalty is assigned if the errors are of the same type, if the same error occurs on the same line and whether the source code of the edit location is unchanged. The value of EQ can be 0 or 1. The closer the value of EQ to 1, the more struggles the students have experienced.

Table 1. Sample Compilation Events of a Programming Session

Comp. No.	Time stamp	Event Type	Error Message	Error Location	Filename
1	1:24:53	Fail	; expected	23	Position.java
2	1:25:22	Fail	; expected	28	Position.java
3	1:25:30	Fail	cannot find symbol - variable YPos	28	Position.java
4	1:25:37	Fail	cannot find symbol - variable YPos	28	Position.java
5	1:25:45	Fail	cannot find symbol - variable time	28	Position.java
6	1:25:58	Success		-1	Position.java
7	1:26:04	Fail	unclosed comment	14	Interest.java
8	1:26:58	Fail	cannot find symbol - class Math	9	Interest.java
9	1:27:05	Fail	cannot find symbol - variable XPos	60	Position.java
10	1:27:45	Fail	cannot find symbol - variable XPos	20	Position.java
11	1:27:55	Success		-1	Interest.java
12	1:28:23	Success		-1	Position.java

The first step in the computation of the EQ Score is to construct a tuple of pairings using the compilation events ordered by timestamp. Referring to data presented in Table 1, the following are the compilation pairings: {{1,2}, {2,3}, {3,4}, {4,5}, {5,6}, {6,7}, {7,8}, {8,9}, {9,10}, {10,11} {11,12}}. For the purpose of the illustration of the scoring, we assume the following value for the Same Edit Location decision: {1,2}- No, {2,3}- Yes, {3,4}- No, {4,5}- Yes, {5,6}- Yes, {6,7}- No, {7,8}-Yes, {8,9}- No, {9,10}- Yes, {10,11}- No, {11,12}.

Table 2 shows the total scores, the EQ of each compilation pairings, and the EQ score of the programming session. Event Type, Error Type, Error Location, and Edit Location corresponds to the penalty assigned based on the algorithm presented in Figure 1 of Section 2.1. Taking the first pair from Table 1, and tracing using the flowchart in Figure 1, we verify first if both events end in a syntax error. Since both ends are in error and both have the same type of error, we assign a penalty of 2 and 3, respectively. A penalty of 0 is assigned to Error Location and Edit Location since the errors occur in a different location and the edit location is not the same. Therefore, the total penalty incurred by the pair {1, 2} is 5. To get the EQ of the first pair, we divide the total penalty by 9 which is the maximum possible score for each pair. We repeat these steps for all the other pairs and then get the average EQ. The average of all the pairs is now the EQ score for this programming session which is 0.39. The closer the value of EQ to 1 means most of the compilations made by the student resulted in errors.

Table 2. EQ Score of the Programming Session

Pairs	Event Type	Error Type	Error Location	Edit Location	Total	EQ
{1,2}	2	3	0	0	5	0.55
{2,3}	2	0	3	1	6	0.66
{3,4}	2	3	3	0	8	0.88
{4,5}	2	3	3	1	9	1
{5,6}	0	0	0	0	0	0
{6,7}	0	0	0	0	0	0
{7,8}	2	0	0	1	3	0.33
{8,9}	2	0	0	0	2	0.22
{9,10}	2	3	0	1	6	0.66
{10,11}	0	0	0	0	0	0
{11,12}	0	0	0	0	0	0
EQ Score						**0.39**

Calculation of the Watwin Score

Watwin Score takes into account the source files students are working with and the time spent by the students in resolving the errors in their program as compared to the resolve times of their peers. Similar to EQ, Watwin Score has a range of values from 0 to 1.

To compute for the Watwin score, we need to construct first a tuple of pairings using the compilation events associated with a file, ordered according to timestamp. Assuming that there are no identical code snapshots and no commented and deletion fixes were done in the source code, the following are the tuple of pairings created from Table 1 for the Position.java file {{1,2}, {2,3}, {3,4}, {4,5}, {5,6}, {6,9}, {9,10}, {10,12}} and {{7,8}, {8,11}} for the Interest.java file. Since compilation pair {6,9} has event type pair {SUCCESS, FAIL}, this pair would be excluded in the computation. After pair construction, the generalization of error messages through the removal of all identifier information within each compilation event pairing was also conducted. The error types presented in the study of [6] were used as a reference in the generalization of the error process. The next step is to calculate the time spent by the student in correcting their errors by getting the timestamp difference of the pairs. In cases where the student switches from one file to another, we compute the difference between the timestamp of the previous file and the timestamp of the succeeding file. For instance, the pairs {10, 12} and {8, 11} have different files in between the compilation. To compute for the time of pairs {10, 12} and {8, 11}, we compute the difference between the timestamps of the pairs {11,12}, and {10,11}, respectively. Thus, the times spent by the student expressed in seconds for each compilation pairings are 29, 8, 7, 8, 7, 40, 28 for the Position.java file and 54, 10 for Interest.java. After determining the time spent per compilation pairing, calculate the mean of the pairings, the standard deviation, the difference between the mean and the standard deviation, and the mean plus the standard deviation. The mean for this programming session is 21.22, the standard deviation is 17.37, the time which is 1 standard deviation below the mean is 3.85, and the time which is 1 standard deviation higher than the mean is 38.59. Table 3 shows the scores assigned to each pair by applying the scoring algorithm as discussed in [16].

Table 3. Watwin Score of the Programming Session

Pairs	Error Message	Error Type	Error Location	Resolve Time	Total	Wat win
{1,2}	4	4	0	15	23	0.65
{2,3}	0	0	2	15	17	0.48
{3,4}	4	4	2	15	25	0.71
{4,5}	0	4	2	15	21	0.60
{5,6}	0	0	0	15	15	0.42
{9,10}	4	4	0	25	33	0.94
{10,12}	0	0	0	15	15	0.42
{7,8}	0	0	0	25	25	0.71
{8,11}	0	0	0	15	15	0.42
Watwin Score						**0.59**

Table 3 shows the scores assigned for each compilation pair in a programming session. Error Message corresponds to the penalty for the same full error message, Error Type for the penalty of same error type, Error Location is for the same line penalty, and Resolve Time is for the time to resolve error penalty. The data from Table 1 shows that both compilation events 1 and 2 resulted in the same error with the same error type but occurred in a different line, thus a penalty of 4 is assigned to both Error Message and Error Type while Error Location has no penalty. The next step in the algorithm is to get the Watwin score of the first pair {1, 2} by comparing the time spent by the student in correcting the error with the time which is 1 standard deviation below the mean or the time which is 1 standard deviation above the mean. We recall that the time spent is 29 seconds which is obviously not less than 3.85 and not higher than 38.59 thus, a penalty of 15 is assigned to Resolve Time. The Watwin of the compilation pair is normalized by dividing the total by 35 which is the maximum score for a compilation pair. The Watwin score of 0.59 is calculated by getting the mean of all the Watwin scores of the compilation pairs for the programming session.

Calculation of the RED Score

Repeated Error Density is also a data-driven metric and is used to quantify errors by looking at the number of repeated error strings a student encounters, and the length of these strings. Unlike EQ and Watwin, the value of RED can be greater than or equal to 0.

The computation of the RED score is simpler compared to the calculation of EQ and Watwin Score. To compute for the RED score, we look into the sequences of compilation events where there are strings of repeated errors. Errors that do not constitute repeated error strings would have a RED of 0, while repeated error strings were calculated by using the equation $r_i^2/(r_i + 1)$. Compute the total score for all the sequence of compilation events to get the RED score as shown in Table 4.

Table 4. RED Score of the Programming Session

Compilation Events	Error Message	Repeated Error (r_i)	RED
1, 2	; expected	1	0.5
3, 4, 5	cannot find symbol - variable	2	1.33
6	No error	0	0
7	unclosed comment	0	0
8	cannot find symbol – class	0	0
9, 10	cannot find symbol - variable	1	0.5
11,12	No error	0	0
		RED Score	**3.16**

In the table, compilation events 1 and 2 has; expected error for both compilation events. In this case, there is 1 repeated error string for the two sequences which is given a RED of 0.5. However, for compilation events 3, 4, 5 with error message cannot find symbol - variable, higher RED is given since there are 2 repeated error strings. Hence, the longer the repeated error strings, the higher penalty is assigned which indicates that the student struggles more on that error. Compilation events 6-8 and 11-12 were assigned a RED of 0 since they do not constitute repeated error strings. The RED score for this programming session is 3.16.

RESULTS AND DISCUSSION

The data that were gathered during the First Semester of the School Year 2013-2014 from students enrolled in CS21A - Introduction to Computing 1 allowed us to capture their compilation behaviors. Because of the exclusion of some records, only the compilation records of the 42 students (33 male and 9 female) out of the 80 students who participated in the study were retained for analysis.

We applied the scoring algorithms presented in Section 2 to the compilation records of the 42 students to calculate the EQ, Watwin, and RED scores. The first step for calculating the scores requires the construction of compilation event pairings of the programming session. After the pair construction, the EQ and RED scores can then be calculated. However, additional data preparation processes were done to the data for the calculation of the Watwin scores. The comments in the code snapshots were removed and each compilation event pairs were compared. All pairs with identical snapshots were removed so that the total number of compilation pairings would not be inflated. In addition, the pairings where the first event represents successful compilation were also removed. The next step after pair pruning would be to determine the diff ratio between the pairs of code snapshots. If there were no insertion of additional code to resolve the errors and only deletion fixes were done, then the pairs were also removed. Further, the generalization of error messages through the removal of all identifier information within each compilation event pairing was also conducted. The error types presented in the study of [6] were used as a reference in the generalization of the error process. The final step in the preparation of data for calculation was to estimate the amount of time a student spent working on each compilation pairing. This was done by computing the difference between timestamps of the pairings on a per-file basis. The mean and standard deviation of the programming session of each student were also computed which would be later used to determine the corresponding time penalty. When these processes were done, the algorithm for quantifying the Watwin score was then applied to the data.

To determine the predictive power of the three metrics, linear regression models were built based on EQ, Watwin, and RED scores and the students' performance in class based on the students' midterm exam scores. Before conducting the linear regression analysis, the variables need to satisfy some assumptions to get a valid result. First, an inspection of the scatterplot was conducted to check if there is a linear relationship between the students' EQ, Watwin, and RED scores and students' performance. The variables were also subjected to Casewise diagnostic to check if there were no significant outliers present. Then we confirmed the residual independence using the Durbin-Watson statistic. Residual independence was confirmed by the Durbin-Watson statistic (1.79) for EQ, (1.92) for Watwin, and (1.80) for RED. Furthermore, we checked that the residuals of the regression line are approximately distributed through inspection of a histogram and P-P plot. All of the metrics' variables satisfied all the tests for assumptions, therefore linear regression was performed.

The findings of the linear regression established that EQ could significantly predict students' midterm exam score, F(1,40) = 11.409, p = .002, and accounted for 20.2% of the explained variability in midterm exam score. The regression equation was Midterm Exam Score = 80.422 - 47.881 * EQ. Watwin on the other hand, could also significantly predict students' midterm exam score, F(1,40) = 6.372, p = .016 and accounted for 11.6% of the explained variability in midterm exam score with a regression equation: Midterm Exam Score = 85.966 - 42.197*WatwinScore. We also found that a linear regression established that the students' RED score could significantly predict students' midterm exam score, F(1,40) = 6.701, p = .013, and accounted for 12.2% of the explained variability in midterm exam score. The regression equation was: Midterm Exam Score = 74.248 - 1.172 * RED.

The result of this study provided information on the ability of the metrics to predict students' midterm exam score. Previous studies that were conducted which compare the predictive power of EQ and Watwin reported that Watwin is a stronger predictor than EQ. However, the result of this study revealed that EQ performed better among the three data-driven metrics based on the dataset used and the population considered in this paper. Furthermore, the new metric RED has also outperformed Watwin in predicting the students' midterm exam score.

CONCLUSION AND RECOMMENDATION

EQ and Watwin have been evaluated by other researchers for their predictive powers. However, there are only a few studies conducted to compare these metrics and there are some inconsistencies in the results of previous researches. In this paper, we compared the predictive power of the three metrics. The result of this study is inconsistent with the results of the previous studies wherein they have reported that Watwin is a better predictor than EQ when correlated with student performance. The result showed that EQ is a better predictor among the three metrics. This result suggests further comparison and exploration of the three metrics in different contexts to come up with consistent findings regarding their ability to predict students' programming performance. And since this is the first time that RED was correlated with students' performance, further validation of the predictive power of RED should be conducted to provide additional evidence that this metric could be considered to be predictive of students' programming performance.

For future works, further comparison of the three metrics using different datasets would be conducted to validate the result of this study.

REFERENCES

[1] Kofi Adu-Manusarpong, John Kingsley Arthur, and Prince Yaw Owusu Amoako. 2013. Causes of Failure of Students in Computer Programming Courses: The Teacher Learner Perspective. International Journal of Computer Applications IJCA 77, 12 (2013), 27–32. DOI:http://dx.doi.org/10.5120/13448-1311

[2] Alireza Ahadi, Raymond Lister, and Arto Vihavainen. 2016. On the Number of Attempts Students Made on Some Online Programming Exercises During Semester and their Subsequent Performance on Final Exam Questions. ITiCSE '16, July 09-13, 2016, Arequipa, Peru. DOI: http://dx.doi.org/10.1145/2899415.2899452

[3] 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 (ITiCSE '16). ACM, New York, NY, USA, 296-301. DOI: http://dx.doi.org/10.1145/2899415.2899463

[4] Adam Carter, Christopher Hundhausen, and Olusola Adesope. 2015. The Normalized Programming State Model: Predicting Student Performance in Computing Courses Based on Programming Behavior. ICER '15, August 9--13, 2015, Omaha, Nebraska, USA. DOI: http://dx.doi.org/10.1145/2787622.2787710

[5] Cruz Izu, Amali Weerasinghe, and Cheryl Pope. 2016. A Study of Code Design Skills in Novice Programmers using the SOLO taxonomy. In Proceedings of the 2016 ACM Conference on International Computing Education Research (ICER '16). ACM, New

York, NY, USA, 251-259. DOI: http://dx.doi.org/10.1145/2960310.2960324

[6] Matthew Jadud. 2005. A First Look at Novice Compilation Behaviour Using BlueJ. Computer Science Education, 15, 1, 25-40, DOI: 10.1080/08993400500056530

[7] Matthew Jadud. 2006. An Exploration of Novice Compilation Behavior in BlueJ. PhD Dissertation, 2006.

[8] Matthew Jadud. 2006. Methods and tools for exploring novice compilation behaviour. In Proceedings of the Second International Workshop on Computing Education Research, ICER '06, pages 73–84, 2006.

[9] Theodora Koulouri, Stanislao Lauria, and Robert Macredie. 2014. Teaching Introductory Programming: A Quantitative Evaluation of Different Approaches. Trans. Comput. Educ. 14, 4, Article 26 (December 2014), 28 pages. DOI=http://dx.doi.org/10.1145/2662412

[10] Craig Miller, Amber Settle, and John Lalor. 2015. Learning Object-Oriented Programming in Python: Towards an Inventory of Difficulties and Testing Pitfalls. In Proceedings of the 16th Annual Conference on Information Technology Education (SIGITE '15). ACM, New York, NY, USA, 59-64. DOI: http://rizal.lib.admu.edu.ph:2134/10.1145/2808006.2808017

[11] 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 (Koli Calling '15). ACM, New York, NY, USA, 77-86. DOI: http://dx.doi.org/10.1145/2828959.2828966

[12] Ma. Mercedes T. Rodrigo, Emily Tabanao, Ma. Beatriz Lahoz, and Matthew Jadud. 2009. Analyzing Online Protocols to Characterize Novice Java Programmers. Philippine Journal of Science. 138 (2): 177-190, December 2009 ISSN 0031 – 7683

[13] Emily Tabanao, Ma. Mercedes T. Rodrigo, and Matthew Jadud. 2011. 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 (2011). DOI: http://dx.doi.org/10.1145/2016911.2016930

[14] Christopher Watson and Frederick Li. 2014. Failure rates in introductory programming revisited. In Proceedings of the 2014 conference on Innovation technology in computer science education (ITiCSE '14). New York: Association for Computing Machinery (ACM), pp. 39-44.

[15] Christopher Watson, Frederick Li, and Jaime Godwin. 2014. No Tests Required: Comparing Traditional and Dynamic Predictors of Programming Success. SIGCSE '14, March 5–8, 2014, Atlanta, GA, USA. http://dx.doi.org/10.1145/2538862.2538930

[16] Christopher Watson, Frederick Li, and Jaime Godwin. 2014. Predicting Performance in an Introductory Programming Course by Logging and Analyzing Student Programming Behavior. 2013 IEEE 13th International Conference on Advanced Learning Technologies. DOI 10.1109/ICALT.2013.99