Predicting At-Risk Novice Java Programmers Through the Analysis of Online Protocols

Emily S. Tabanao
Department of Information Technology
MSU-Iligan Institute of Technology
Iligan City, Philippines
emily.tabanao@g.msuiit.edu.ph

Ma. Mercedes T. Rodrigo
Department of Information Systems and Computer Science
Ateneo de Manila University
Quezon City, Philippines
mrodrigo@ateneo.edu

Matthew C. Jadud
Department of Computer Science
Allegheny College
Meadville, PA, USA
matthew.c@jadud.com

ABSTRACT

In this study, we attempted to quantify indicators of novice programmer progress in the task of writing programs,and we evaluated the use of these indicators for identifying academically at-risk students. Over the course of nine weeks, students completed five different graded programming exercises in a computer lab. Using an instrumented version of BlueJ, an integrated development environment for Java, we collected novice compilations and explored the errors novices encountered, the locations of these errors, and the frequency with which novices compiled their programs.We identified which frequently encountered errors and which compilation behaviorswere characteristic of at-risk students. Based on these findings, we developed linear regression models that allowed prediction of scores on a midterm exam. However, the models derived could not accurately predict the at-risk students. Although our goal of identifying at-risk students was not attained, we have gained insightsregarding the compilation behavior of our students, which may help us identify students who are in need of intervention.

Categories and Subject Descriptors

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

General Terms

Human Factors

Keywords

Novice Programmers, Achievement, Compilation Behavior, Java programming, CS1

1. INTRODUCTION

Learning to program is hard. In a study conducted by McCracken et al., researchers found that as many as 35% of students fail their first programming course, while in the United Kingdom and the United States approximately 30% of computer science students did not understand programming basics [18]. This has led to a growing concern among computer science educators over the lack of programming comprehension of first-year computer science students.

Novice computer programmers are of special interest to the computer science education community. Computer science education researchers have conducted investigations into the kinds of problems novice programmers encounter when learning to program. has been found to be limited and shallow, hence they struggle with writing syntactically-correct programs [18]. They typically lack detailed mental models of various programming constructs [4,9], and tendto organize knowledge based on superficial similarities. Novices use general problem solving strategies instead of problem specific or program specific strategies, and approachprogramming "line by line" rather than at the level of meaningful program structures [21,25]. Novices have been observed to have poor program comprehension as evidenced by activities involving the tracing of their code [19]. They had a poor grasp at how a program executes and have problems with understanding that each instruction is executed in the state that has been created by the previous instructions [1].

Over the years, computer science educators and researchers have also conducted studies to identify the causes of these problems and to find possible solutions. One such method used is the collection and analysis of online protocols, which typically involve gathering information by augmenting the programming environments students use to write, compile and test their programs, thus allowing for automated data collection and subsequent analysis. In this study,
interactions with the Java compiler (mediated by the BlueJ IDE), which include the error message, the line number where the error appeared, the source code, and the time the compilation occurred.

The goal of this study was to determine whether at-risk novice Java programmers can be accurately identified through the analysis of online protocols. We quantified the compilation behavior of the students by computing what Jadud calls the Error Quotient (Section 2.1) [14]. We also derived the errors encountered and time between compilations from the data logs. From this, we built a linear regression model based on the EQ, errors encountered and time between compilations and the midterm exam scores.

This present study has several potential benefits to educators teaching programming. Identifying at-risk students early in the semester can help educators provide targeted help and proper intervention to those who need it the most. By knowing the errors students typically incur in their programming, educators can better address concepts that students find difficult to grasp or are having misconceptions about. The EQ score can tell who among the students are struggling with syntax errors, prompting teachers to intervene to mitigate frustration. Spotting at-risk students early may also help reduce the dropout rates in computer science classes.

1.1 Research questions

From the online protocols of our students, we ask the following questions:

1. How do students with different achievement levels differ in terms of:
    a. Error profiles?
    b. Average time between compilation profiles?
    c. EQ profiles?
2. What factors predict the midterm scores?

2. REVIEW OF RELATED LITERATURE

Many studies have been conducted that identified factors related to success in the learning of programming. Among the identified student characteristics that may contribute to student success in introductory programming courses are prior programming experience, gender, secondary school performance and dislike of programming, intrinsic motivation and comfort level, high school mathematics background, attribution to luck for success/failure, formal training in programming, and perceived understanding of the material [5, 11, 12, 24].

Other studies have investigated teaching and learning approaches in relation to success in learning programming. Byckling and Sajaniemi developed the concept of *roles of variables* and a visualization tool to support the concept. Initial experiments with the visualization of variable roles helped novices in their learning of programming [7]. A subsequent experiment that tested the tool showed that visualization significantly improved the debugging skill of the students [2].

Mayer stated that mental models are crucial to learning and understanding programming [8]. To this end, Dehnadi designed an instrument to reveal the learners' mental models regarding assignment of values to variables and to evaluate the consistency and viability of those mental models. He believed that consistency of mental model is more important than viability of the model. That is, if the learner believes there is one and only one rule that applies, and consistently applies that rule, then the consistent application of the rule can have a strong effect on success in early learning of programming [8]. However, after several experiments the instrument failed to accurately deliver expected results [6].

It is generally accepted that programming is not easy. When novices were interviewed about their experience while performing their programming assignments, students clearly recalled emotional experiences and reactions. One of the most harmful of these emotions is frustration [16]. Some students learned from the bugs they encounter in their programs, but others get frustrated every time they encounter a problem and tend to view bugs as a measure of their performance. Labeled by Perkins as *stoppers*, these students have a tendency to get discouraged by their mistakes and give up [19]. Recently, Rodrigo & Baker was able to build a model that predicted frustration from the online protocols of the students [22]. In another study, Rodrigo et al. found confusion, boredom and IDE-related on-task conversation had negative effect on achievement in an intro to programming course [23].

Other attitude and behavior that were found to have positive effect on achievement in early programming course are perfectionism and self-esteem, and high states of arousal or delight [15]. Negative attitude such as disliking programming was found to be associated with lower success in early programming courses [3].

2.1 Error Quotient

In an attempt to represent student compilation behavior as a single scalar quantity, Jadud developed the Error Quotient (EQ). EQ is a function of error type, location, and proximity in time relative to other errors. It was intended as an indicator of how well or poorly a student was progressing.

Every record in the online protocol represents one compilation event. Stored in each record is the error message (if there was an error at the time of compilation), the location (line number) of the error in the file, and the source code. To compute the EQ, we examine the error message, the line number and the text of the source code. Given two compilation events, we first check whether both compilations ended in error. If they did, we assign a penalty. We then compare the error messages encountered. If they are the same, another penalty is imposed. If the errors occurred on the same line numbers, a third penalty is imposed. Finally, the programmer incurs a fourth penalty if the edit location of the source code on both events are the same. The penalties are normalized and averaged across all pairs of compilations to arrive at the final EQ score of the session.

An EQ score ranges from 0 to 1.0, where 0 is a perfect score. An EQ score of 0 means that at no point did the student encounter errors in consecutive compilations. A score of 1.0 means that every compilation resulted to the same syntax error in the same location.

3. METHODOLOGY

This study was conducted in the Department of Information Systems and Computer Science (DISCS) of the Ateneo de Manila University on the First Semester of School Year 2007-2008. The course *CS21A - Introduction to Computing I* (CS1 in the literature) is the first computer programming course offered by the department to students studying Computer Science or Management Information System, and it is a required course for students in both degree programs. The Computer Science students take the course during their first year in the program, while the MIS students take it during their second year. It is presumed that students taking the course do not have prior knowledge of programming, but it is expected that they know basic operations of how to use a computer.

The participants of this study were the students enrolled in CS21A on the first semester of 2007-2008 at the Ateneo de Manila University. Of the 143 subjects participating, 35% were female and 65% were male; 18% were students in the Computer Science program, and 82% were students in Management Information Systems.

On the first day of classes students were informed about the study. A consent letter was given to each of the students. Each student affixed their signature and the signature of their parent/guardian if they were willing to be part of the study. They were not, however, obliged to join the study.

The students perform laboratory exercises in the computing laboratories that also served as lecture rooms. All the machines were installed with the same operating system, Java standard development kit, and BlueJ. The machines were connected to a local area network and the Internet. The laboratories have one-to-one student-to-computer ratios.

Over the first nine weeks of the semester, we scheduled five days during which students performed laboratory exercises. The lab exercises followed the lectures of the topics covered in the class and were designed to be finished in a one hour laboratory session. The exercises were given to the students on the day of their scheduled laboratory. A driver program was also given in each exercise for the students to test their code. Data gathering of the compilation logs was completely automated. Data was gathered only on the scheduled laboratory when the standard exercises were given.

3.1 Tools for collecting data

The data gathering tool was implemented as an extension to the BlueJ programming environment [14]. The focus of this study was on the compilation activities of the students inside the BlueJ IDE. A compilation event data was captured every time a student clicked the Compile button. Compilation events were recorded and stored in an SQlite database. Each record in the database represents one compilation event. We only retained data for students who consented to participate in the study.

3.2 Data pre-processing

Once the data had been gathered, it was pre-processed for analysis. For each study participant, all compilation records not related to the lab exercise were deleted. Students whose online protocol records were incomplete because of absences or technical problems were deleted from the dataset as well. After all the deletions, the sample was reduced from 143 to 124 students; 79 were male and 45 were female.

4. RESULTS AND DISCUSSION

The tools that we used for data collection allowed us to capture a copy of the students' work in progress every time they compiled their code. Here we present the errors encountered, the time between compilations, the computed Error Quotients, and linear regression modeling results. Computed results for the entire five lab sessions will be presented.

Using the midterm exam score, we categorized our students into three groups to determine behaviors of different student categories. Students who were one standard deviation below the mean were called the AtRisk group. Those who were one standard deviation above the mean were the HighPerforming group and those who were within one standard deviation from the mean are the Average group. Twenty-three out of 124 students were in the AtRisk group. They received a score of 62 or below. Twenty-five out of 124 students were in the HighPerforming group. They received scores of 89 and above. The Average group was composed of 76 students.

For each student, we extracted the timestamps, the errors encountered if there were any together with the line number and the contents of the file compiled. We computed the average time between compilations, total compilations, the EQ and the sum of all errors encountered for the five lab sessions. We also made a tally of the frequency of compilations per ten second bins and the top ten errors encountered.

A total of 24,151 compilation events were collected during the five laboratory sessions, 14,470 of these events or 60% ended in error. Table 1 shows the breakdown of these compilation events among the three student groups. Notice that the AtRisk group incurred the highest percentage of errors among the three groups even if they only made up 19% of our participants.

Table 1. Total Compilation Events per student group

Student Group	Total Compilation Events	Events with Errors	Percentage of events with errors
AtRisk	4,822	3,169	66%
Average	15,720	9,532	61%
HighPerforming	3,609	1,769	49%
Total	24,151	14,470	60%

4.1.a. Do different groups of students have different error profiles?

Using the total error events, we compared the groups using a one-way ANOVA to determine whether there were significant differences between the groups in terms of the errors they encountered. When the F test was significant, we proceeded with post-hoc analysis using Tukey HSD to check which among the groups had significant differences.

There was a significant difference among groups on the Total Errors, $F(2,121)=8.97$, $p<.001$. Post-hoc test results showed that the HighPerforming group had a significant lower number of errors encountered from the AtRisk group at $p < .001$. The HighPerforming group also had a significant lower number of errors compared to the Average group at $p < .01$, and the Average group was not significantly different from the AtRisk group. Post-hoc test results also revealed that students in the AtRisk group encountered the most errors among the three groups.

We were also interested to know what kinds of compilation errors each group encountered most frequently. We disaggregated the total errors into the top ten errors incurred by all the subjects. Figure 1 shows the top ten errors encountered over five lab sessions broken down by group.

We then performed a one-way ANOVA on each error type to determine whether the incidence of these errors among the different groups were significantly different. For the errors where significant differences were found, another post-hoc test was performed. ANOVA revealed five errors where significant differences between the three groups were found.

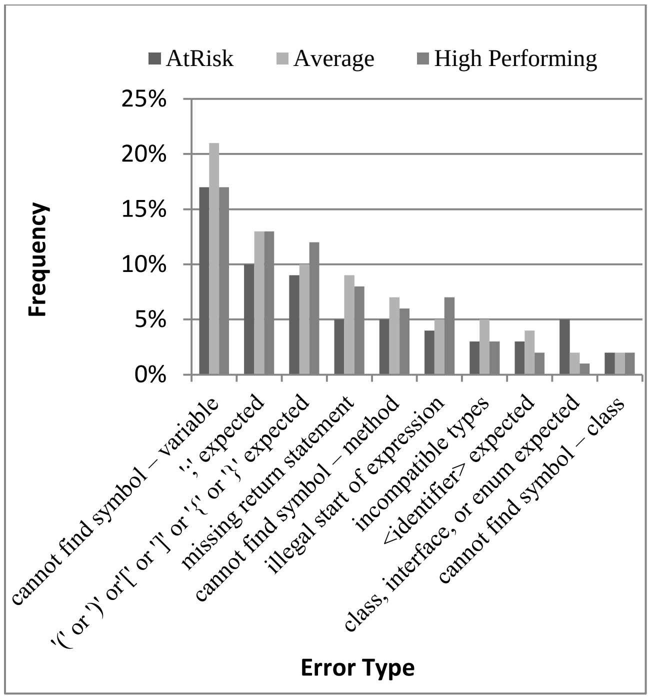

Figure 1. Top Ten Errors over Five Lab Sessions broken down by Student Group

Table 2 shows the results of the post-hoc test on these errors.

Table 2. Tukey HSD test on Error Types with Significant F test at 95% confidence interval

Error Type	Groups Compared	p adj
`cannot find symbol-variable`	Average-AtRisk	0.02
	HighPerforming-AtRisk	0.00
	HighPerforming-Average	0.01
`incompatible types`	Average-AtRisk	0.91
	HighPerforming-AtRisk	0.08
	HighPerforming-Average	0.06
`identifier expected`	Average-AtRisk	0.33
	HighPerforming-AtRisk	0.00
	HighPerforming-Average	0.01
`class, interface, or enum expected`	Average-AtRisk	0.00
	HighPerforming-AtRisk	0.00
	HighPerforming-Average	0.70
`cannot find symbol-class`	Average-AtRisk	0.68
	HighPerforming-AtRisk	0.04
	HighPerforming-Average	0.09

Table 2 shows that:

1. the HighPerforming group was significantly different from the AtRisk group on only four errors: `cannot find symbol-variable`, `identifier expected`, `class, interface, or enum expected`, and `cannot find symbol-class` errors. The mean differences show that the HighPerforming group encountered fewer instances of these errors compared to the AtRisk group;
2. the HighPerforming group was significantly different from the Average group on only two errors the `cannot find symbol-variable` and `identifier expected` errors. The HighPerforming group encountered fewer instances of these two errors as compared to the Average group; and
3. the Average group was significantly different from the AtRisk group on only two errors the `cannot find symbol-variable` and `class, interface, or enum expected` errors. The AtRisk group encountered more instances of these two errors when compared with the Average group.

4.1.b. Do different groups have different average time between compilations profiles?

Every time students compiled their code, our data collection program recorded a timestamp, the time the event happened. Timestamps were precise up to the millisecond level. We did not attempt to categorize students based on the precise time between compilations this level of granularity was too fine-grained for our purposes. Instead, a
time between compilations into 10 second bins and performed a one-way ANOVA on the three student groups to determine if there were differences in their compilation behavior in terms of how rapidly they compiled their programs. Figure 2 shows a comparison of the compilation of the groups of students.

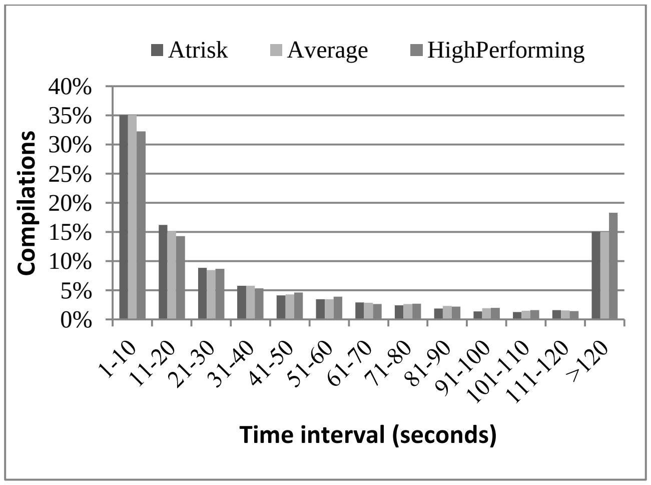

Figure 2. Time between compilations over Five Lab Sessions broken by Student Group

Table 3 summarizes the time interval in seconds at which there was a significant difference among the three student groups. Further analysis using Tukey HSD was performed and results are also shown in Table 3, from which we can see that:

1. there was no significant difference between the Average and AtRisk group in terms of time between compilations;
2. there was a significant difference between the HighPerforming and the Average group except on the time intervals 21-30, 111-120 and >120 seconds. The HighPerforming group performed fewer compilations in all time intervals compared to the Average group; and
3. the HighPerforming group was significantly different from the AtRisk group except on the time interval 81-90 seconds.

The AtRisk group performed more compilations in all time intervals compared to the HighPerforming group.

Table 3. Tukey HSD test on time intervals with significant F test at 95% confidence interval

Time interval (seconds)	Groups Compared	p adj
1-10	Average-AtRisk	0.71
	HighPerforming-AtRisk	0.04
	HighPerforming-Average	0.05
11-20	Average-AtRisk	0.25
	HighPerforming-AtRisk	0.00
	HighPerforming-Average	0.01
21-30	Average-AtRisk	0.35
	HighPerforming-AtRisk	0.01
	HighPerforming-Average	0.06
31-40	Average-AtRisk	0.53
	HighPerforming-AtRisk	0.00
	HighPerforming-Average	0.01
61-70	Average-AtRisk	0.66
	HighPerforming-AtRisk	0.02
	HighPerforming-Average	0.04
81-90	Average-AtRisk	0.81
	HighPerforming-AtRisk	0.31
	HighPerforming-Average	0.04
111-120	Average-AtRisk	0.53
	HighPerforming-AtRisk	0.02
	HighPerforming-Average	0.10
>120	Average-AtRisk	0.21
	HighPerforming-AtRisk	0.01
	HighPerforming-Average	0.10

These results suggest that we cannot differentiate an Average from an AtRisk student based on the timing of when they click the compile button. The HighPerforming group, on the other hand, can always be differentiated from the Average and the AtRisk groups. Though there is one time interval where the HighPerforming group is not significantly different from the AtRisk group, particularly the 81-90 seconds interval, overall, the difference is significant at p <.05. There is a need to further examine the data to check what the AtRisk group was doing exactly. We suspected they could be doing something not related to the assignment or maybe they seemed lost or got stuck and did not know what to do as uncovered by the study of [16]. It is possible that although the AtRisk group spends a significant amount of time between compilations, the time between compilations is not spent productively.

4.1.c. Do different groups of students have different EQ Profiles?

We took the average Error Quotient (EQ) of the students for the five lab sessions and used as our data for this analysis. We wanted to see how different the EQ scores were of the three student groups. Figure 3 shows the EQ scores and percentage distribution among our participants.

Result of one-way ANOVA showed that there was a significant difference in the EQ scores among the groups, F(2,121) = 20.528, p < .001. After performing post-hoc analysis, we found that the Average group is significantly different from the AtRisk group at

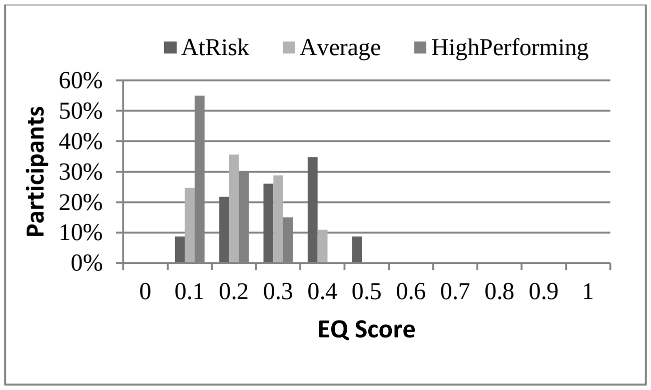

Figure 3. EQ Score distribution by Group

p< .001 and the mean difference show that the AtRisk group have higher EQs compared to the Average group. The HighPerforming group was significantly different from the Average group at p<.01 and the mean difference show that the HighPerforming group have lower EQs compared to the Average group. The AtRisk group was significantly different from the HighPerforming group at p<.001 and the mean difference show that the AtRisk group had higher EQs compared to the HighPerforming group (Table 4).

Table 4. Tukey HSD test on EQ at 95% confidence interval

Groups Compared	Mean difference	p adj
Average-AtRisk	-0.11	0.00
HighPerforming-AtRisk	-0.19	0.00
HighPerforming-Average	-0.08	0.00

4.2. What factors predict the midterm score?

In this part of our data analysis, we first correlated the Total Errors encountered and average time between compilations with the Midterm Score. It was important to get a correlation from these variables first in order to see if there was any relationship between them. If there was no relationship existed between them, linear regression would fail.Pearson's product-moment correlation showed the following results:

1. there was a significant negative relationship between the total errors encountered and the Midterm Score, r = -.41, p<.001;
2. there was a significant positive relationship between average time between compilations and the Midterm Score, r = .27, p< .01; and
3. there was a significant negative relationship between EQ and the Midterm Score, r= -.55, p <.001.

Results revealed that there was a correlation between the errors encountered, average time between compilations and the Error Quotient. We then proceeded with performing linear regression to determine whether any of these were predictors of achievement as
To test
the generalizability of our models we also computed
Bayesian Information Criterion for . The
is used to assess the tradeoff between model fit and the number of parameters (which can spuriously increase model fit).
-6 signify that the model has significantly better fit than chance, given the number of model parameters [20].

4.2.a. Predicting midterm score using the errors encountered

Using the Total Errors encountered from the five lab sessions and the Midterm Score we arrived at the following model:

Model 1: MidtermScore = 84.29698 - 0.07304*TotalErrors

Multiple R^2=0.1457, Adjusted R^2=0.1387
F(1,122)=20.81, p-value<.001, BIC = -7.8

Model 1 accounted for only 13.87% of the variance of the Midterm score. From the intercept of Model 1, we concluded that the model could not predict the HighPerforming group. However, this could also mean that the errors encountered by the HighPerforming group of students did not affect their MidtermScore.

We performed multiple regression by inputting the top ten errors into our modeling tool. Only three out of the ten errors were significant as seen in Model 2.

Model 2:
MidtermScore = 83.50274-0.25632*UNKNOWN_VARIABLE - 0.42035*CLASS_INTERFACE_EXP
- 0.75506*UNKNOWN_CLASS

Multiple R^2=0.2645, Adjusted R^2=0.1994,F(10,113)=4.063, p-
-10.2635

Model 2 seemed to apply only to the Average and the AtRisk groups. It could not predict HighPerforming students but it is much better compared to Model 1 in terms of predicting the AtRisk students.

From the results in Section 4.1.a, we found that among the top 10 errors, there were only five errors where the three groups had significant differences. And out of the five, three came out in Model 2 that significantly affects the Midterm Score. As added information and to help us further understand Model 2, we generated the means of the error types per group. Table 5 shows the computed means.

Table 5. Group mean values on error types

Error Type	Mean Values		
	High-Performing	Ave-rage	AtRisk
`cannot find symbol – variable`	12.0	23.3	33.6
`incompatible types`	1.3	5.6	6.3
`identifier expected`	1.9	4.2	5.6
`class, interface, or enum expected`	0.8	2.2	7.8
`cannot find symbol-class`	1.1	2.7	3.3

Table 5 shows that the HighPerforming group incurred the least number of errors, which
Midterm Score was not affected by the errors they encountered. However, the Midterm Scores of the students in the Average and AtRisk group may be negatively affected by the following errors: `cannot find symbol–variable, class, interface, or enum expected` and `cannot find symbol-class`. Model 2 indicates that the more `cannot find symbol – variable, class, interface, or enum expected` and `cannot find symbol-class` errors the AtRisk and Average group encounters, the lower their midterm scores will be. Also, we could notice in Table 5 that the AtRisk and Average groups had closer means and were higher compared to the HighPerforming group.

4.2.b. Predicting the midterm score using Time between compilations

To determine if time between compilations was a good predictor of the Midterm score, we took the mean of the average time between compilations of the five lab sessions and came up with Model 3.

Model 3: MidtermScore = 65.04788
+ 0.12107*AverageTBC_seconds

Multiple R^2=0.07272, Adjusted R^2=0.06512
F(1,122)=9.568, p-value<.01, BIC = -1.97243

Model 3 suggests that average time between compilations had a positive effect on the Midterm score. The longer the time intervals between compilations, the higher the Midterm score will be. Although this finding was intuitive, the model itself was quite weak at an R^2 -1.97. When we computed the predicted Midterm Scores the model was not able to predict AtRisk students and all students were predicted to belong to only one group.

4.2.c. Predicting midterm score using EQ scores

We computed the average EQ score for the five laboratory sessions and together with the Midterm Score arrived at Model 4.

Model 4: MidtermScore = 92.918 - 64.396*EQ

Adjusted R^2 = 0.2971, F(1,122) = 52.98,
p-value = 3.591e- -17.3303

In all the models that we have, Model 4 has the highest R^2 value. The BiC
than chance given the EQ parameter. Model 4 suggests that EQ has a negative effect on the Midterm score. The higher the EQ, the lower the Midterm Score will be.

We combined all factors in Models 1 to 4 to see if we can come up with a better model. These factors are the errors `UNKNOWN_VARIABLE`, `CLASS_INTERFACE_EXP` and `UNKNOWN_CLASS` and the EQ. We arrived at Model 5. The EQ came out to be a highly significant predictor of the Midterm score.

Model 5: MidtermScore = 90.58643-43.33380*EQ

Adjusted R^2 = 0.3073, F(7,116)= 8.795,
p-value=1.202e- -20.8326

The `UNKNOWN_VARIABLE` error had p=.06, an indication to look for in determining AtRisk students. We can see from Model 5 that the EQ accounted for about one-third of the variance of the Midterm Score based on the adjusted R^2
indicates that our model has a significantly better fit than chance given the parameters.

5. CONCLUSION AND FUTURE WORK

In this paper, we attempted to determine whether there were differences among groups of students in terms of their error profiles, compilation profiles and Error Quotient profiles. We also tried to determine whether any of these indicators could predict the midterm scores. Based upon

we found that errors encountered by novices in Java programming can negatively affect their midterm score. We have shown that High Performing novices encountered similar errors with the rest of their peers. However, when compared against the Average and AtRisk novices, High Performing novices encountered these errors less frequently. The Average and AtRisk novices encountered more errors, and these two groups differed significantly on only three types out of the top ten errors. We found that errors did not affect the Midterm Scores of the HighPerforming novices but there were three types of error that negatively affected the Midterm Scores of the Average and AtRisk novices: `cannot find symbol - variable`, `class, interface, or enum expected` and `cannot find symbol-class` errors.

We also found that higher time between compilations yielded positive effect on the Midterm Score. The average time between compilations performed by the High Performing novices was higher than those of the Average and the AtRisk novices. The HighPerforming group spent more time on their programs and wrote more lines of code compared to the other two groups. There was a significantly high incidence of recompilations in less than 30 seconds among the AtRisk and Average novices. The compilation behavior of the two groups was the same.

We generated linear regression models that predicted the Midterm Score of a student given the errors encountered, time between compilation and the EQ across five lab sessions. The model with EQ as a factor could significantly predict Midterm Scores better than would-be-expected by chance. However, our models came out to be poor at predicting AtRisk students.

From the significant findings of this study, we identify several possible avenues for further investigation. We could use other achievement indicators such the grade for lab exercise where data were collected to strengthen our models. We can also improve on the basis of grouping the students. Instead of just one group to represent the average students, we can break them into high-average and below-average students. We can use the standard cut-off in the grading systems for A, B, C and D. We surmised that high-average students had similar compilation behavior with the high performing students and that below average and at-risk students may have had similar behaviors as well.

The models derived from the errors encountered suggest that we look out on students who struggle with errors. Additional investigations can be undertaken to explore the conceptual errors that generate these syntax errors. The differences in the types of errors encountered by AtRisk and HighPerforming students, for example, may say some

and where their understandings are flawed. Insight into these flaws can help computer science educators design interventions to correct these mental models.

Finally, we believe there is potential to use IDEs intelligent systems that offer help or guidance to students or gives signal to the teacher for provision of help to some students. Support systems of this type may help mitigate student frustration and confusion, increase programming comprehension, and raise achievement. They may also contribute to greater student retention in computer science and related disciplines.

In conclusion, this study has shed light on the differences of the profiles of novice programmers. Using the compilation logs we can spot who among the novices are performing well. Though we have not successfully attained our goal of identifying at-risk students, the study has given us clues on which compilation behaviors at-risk students are exhibiting. Our suggestions on improving the study might give us better models in the future.

6. ACKNOWLEDGMENTS

We thank Arvin Guingguing, Anna Christine Amarra, Ramil Bataller, Andrei Coronel, Darlene Daig, Jose Alfredo de Vera, Thomas Dy, Maria Beatriz Espejo-Lahoz, Dr. Emmanuel Lagare, Sheryl Ann Lim, Ramon Francisco Mejia, Shiela Pascua, Dr. John Paul Vergara, and the technical and secretarial staff of the Ateneo

Science for their assistance with this project. We thank the Ateneo
d -2008, for their participation. We thank the Department of Science and

Technology Research and Development for making this study possible by providing the grants entitled Modeling Novice Programmer Behaviors Through the Analysis of Logged Online Protocols and Observation and Diagnosis of Novice Programmer Skills and Behaviors Using Logged Online Protocols. Finally, Dr. Rodrigo thanks the Philippine American Educational Foundation and the Council for International Exchange of Scholars for her 2008-2009 Advanced Research and University Lecturing Fulbright Scholarship.

7. REFERENCES

[1] Ahmadzadeh, M., Elliman, D. and Higgins, C. An analysis of patterns of debugging among novice computer science students. In Proceedings of the 10th annual SIGCSE conference on Innovation and technology in computer science education (ITiCSE '05). ACM, New York, NY, USA, 84-88. http://doi.acm.org/10.1145/1067445.1067472

[2] Al-Barakati, N., Al-Aama, A., 2009. The effect of visualizing roles of variables on student performance in an introductory programming course. SIGCSE Bull. 41, 3 (July 2009), 228-232. http://doi.acm.org/10.1145/1595496.1562949

[3] Bennedsen, J., Caspersen M. E. 2008. Optimists Have More Fun, But Do They Learn Better? - On the Influence of Emotional and Social Factors on Learning Introductory Computer Science. Computer Science Education, 18, 1, 1-16.

[4] Ben-Ari, M. 1998. Constructivism in computer science education. ACM Press New York, NY, USA. Vol. 30(1).

[5] Bergin, S., Reilly R.: Programming: Factors that influence success. SIGCSE 2005. Proceedings of the thirty-fifth SIGCSE technical symposium on Computer Science Education. St. Louis, Illinois, US. February 2005, 411-415.

[6] Bornat, R., Dehnadi, S., Simon. 2008. Mental models, consistency and programming aptitude. In *Proceedings of the tenth conference on Australasian computing education - Volume 78* (ACE '08), Simon Hamilton and Margaret Hamilton (Eds.), Vol. 78. Australian Computer Society, Inc., Darlinghurst, Australia, Australia, 53-61.

[7] Byckling, P., Sajaniemi, J., 2006. Roles of variables and programming skills improvement. *SIGCSE Bull.* 38, 1 (March 2006), 413-417. DOI=10.1145/1124706.1121470 http://doi.acm.org/10.1145/1124706.1121470

[8] Dehnadi, Saeed. A Cognitive Study of Learning to Program in Introductory Programming Courses [Doctoral Thesis]. Retrieved from http://eprints.mdx.ac.uk/6274/1/Dehnadi_A_Cognitive_Study_of_Learning.pdf

[9] duBoulay, B. 1986. Some difficulties of learning to program. Journal of Educational Computing Research, Vol. 2, pp. 57--73.

[10] Fenwick, J.B.Jr., Norris, C., Barry, F.E., Rountree,J., Spicer,C.J. and Cheek, S.D. 2009. Another look at the behaviors of novice programmers. SIGCSE Bull. 41, 1 (March 2009), 296-300. DOI=10.1145/1539024.1508973 http://doi.acm.org/10.1145/1539024.1508973

[11] Goold, A. and Rimmer, R. (2000): Factors affecting performance in first-year computing. ACM SIGCSE Bulletin, 32(2): 39-43.

[12] Hagan, D., Markham, S., 2000. Does it help to have some programming experience before beginning a computing degree program?.*SIGCSE Bull.* 32, 3 (July 2000), 25-28. http://doi.acm.org/10.1145/353519.343063

[13] Jadud MC. (2005). A first look at novice compilation behavior using BlueJ. Computer Science Education, 15(1), 25-40.

[14] Jadud, M.C., 2006. Methods and tools for exploring novice compilation behaviour. Proceedings of the 2006 international workshop on Computing education research. New York, NY, USA : ACM Press. pp. 73--84.

[15] Khan, I., Hierons, M., Brinkman, W. 2007. Mood independent programming. In *Proceedings of the 14th European conference on Cognitive ergonomics: invent! explore!* (ECCE '07). ACM, New York, NY, USA, 269-272. http://doi.acm.org/10.1145/1362550.1362606

[16] Kinnunen, P., Simon, B. 2010. Experiencing programming assignments in CS1: the emotional toll. In Proceedings of the Sixth international workshop on Computing education research (ICER '10). ACM, New York, NY, USA, 77-86. http://doi.acm.org/10.1145/1839594.1839609

[17] Lahtinen, E., Ala-Mutka, K. and Jarvinen, H.M. 2005. A study of the difficulties of novice programmers. ACM Press. ACM SIGCSE Bulletin. New York, NY, USA. Vol. 37(3), pp. 14--18.

[18] McCracken, M., Almstrum, V., Diaz, D., Guzdial, M., Hagan, D., Kolikant, Y.B., Laxer, C., Thomas, L., Utting, I., and Wilusz, T. 2001. A multi-national, multi-institutional study of assessment of programming skills of first-year CS students. In Working group reports from ITiCSE on Innovation and technology in computer science education (ITiCSE-WGR '01). ACM, New York, NY, USA, 125-180. http://doi.acm.org/10.1145/572133.572137

[19] Perkins,D.N., Hancock, C., Hobbs, R., Martin, F. 1986. Conditions of Learning in Novice Programmers**.** Journal of Educational Computing Research, N. 1, Vol. 2, pp. p37--55.

[20] Raftery, A. E.. Bayesian model selection in social research. *Sociological Methodology*, 25, 111-163, 2003.

[21] Robins, A., Rountree, J. &Rountree, N. 2003. Learning and Teaching Programming: A Review and Discussion. Taylor & Francis, Computer Science Education,13(2), pp. 137-172.

[22] Rodrigo, M.M.T., Baker, R. 2009. Coarse-grained detection of student frustration in an introductory programming course. In *Proceedings of the fifth international workshop on Computing education research workshop* (ICER '09). ACM, New York, NY, USA, 75-80. http://doi.acm.org/10.1145/1584322.1584332

[23] Rodrigo,M.M.T., Baker R.S.J.d., Jadud, M.C., Amarra, A.M., Dy, T., Lahoz,M.B.E., Lim, S.L., Pascua,S.A.M.S., Sugay, J.O., and Tabanao, E.S., 2009. Affective and behavioral predictors of novice programmer achievement. *SIGCSE Bull.* 41, 3 (July 2009), 156-160. http://doi.acm.org/10.1145/1595496.1562929

[24] Wilson, B. (2002): A study of factors promoting success in computer science including gender differences.Computer Science Education, 12(1-2):141-164.

[25] Winslow, L.E. 1996. Programming pedagogy - A psychological overview. SIGCSE Bulletin, 28(3), pp. 17-22.