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

Emily S. Tabanao
Department of Information Systems and Computer Science
Ateneo de Manila University
Quezon City, Phillippines
emilytabanao@yahoo.com

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

Matthew C. Jadud
Franklin W. Olin College of Engineering
Needham, MA, 02492 USA
matthew.c@jadud.com

## ABSTRACT
Learning to write a program is a difficult task. In this study we looked at how students progress as they write their programs. Using an instrumented development environment, we captured the errors students encountered, the location of these errors, and the frequency with which students compiled their program. We then correlated these metrics with the students' midterm exam score. We found that the lower the incidence of syntax errors, the higher is their midterm exam score. We see this line of research as valuable tool that may help identify at-risk students early in their development as programmers.

## Keywords
Novice programmers, Java, computer science education.

## 1. INTRODUCTION
Computer science educators are growing increasingly concerned over the lack of programming comprehension of first-year computer science students. Novice programmers exhibit a variety of disturbing problems. In Australia, as many as 35% of students fail their first programming course. In the United Kingdom and the United States, approximately 30% of computer science students did not understand programming basics by the end of this first class [18]. Studies have found that novice programmers could not write syntactically correct programs even after their first programming course [7, 18, 24]. Novice knowledge was limited and shallow. Students lacked detailed mental models and tended to organize knowledge based on superficial similarities. They failed to apply relevant knowledge. They used general problem solving strategies instead of problem specific or program specific strategies, and approached programming "line by line" rather than at the level of meaningful program structures [25, 33]. Novices were poor at program comprehension and in tracing their code [22]. They had a poor grasp at how a program executes. They also had problems with understanding that each instruction is executed in the state that has been created by the previous instructions [5].

As a response to these findings, Computer Science educators and researchers conducted studies to recognize the characteristics of novice programmers, determine the causes of their problems and to find possible solutions. Various methods have been used in the study of novice programmers in the past.

One of these methods is the collection and analysis of online protocols. Online protocols refer to a sequence of program compilations. They are gathered by enhancing the development environments used by students to write, compile and test their programs to such that each compilation attempt stores the source code and related information—e.g. error message, the line number where the error occurred, and the time the compilation occurred—in a database. These data allow researchers to look into the students' progress as they write their programs, identify common errors and behavior [ 1, 9, 10, 11, 16, 30, 32].

The goal of this study is to determine whether an analysis of online protocols can successfully identify at risk-novice Java programmers. From the online protocols, we quantify the compilation behavior of the student by computing what Jadud calls as the error quotient (EQ) [9]. We also derive the errors encountered and time between compilations from the data logs. We then correlate the EQ, errors encountered and time between compilations to the student's achievement in class, particularly their midterm exam scores. A significant result from this correlation may allow almost accurate identification of at risk students in a novice programming class.

### 1.1 Significance of the Study
This paper would like to answer the following questions: Can online protocols identify and characterize at-risk novice java programmers? Is there a relationship between errors encountered and students' achievement in class? What errors do students commonly encounter? What is the pattern of students' program compilation? Answers to these questions can have several potential benefits in the field of Computer Science Education. Identifying at-risk students early in the semester can help educators provide targeted help and proper intervention to those who need it most. By knowing the errors typically students are making and the time it takes them to fix these errors, educators can better address concepts that students find difficult to grasp or misconceptions. Even a simple error like a missing semi-colon can be very difficult for a novice to correct. An error that takes a large amount of time to correct can be very frustrating. 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 programming classes.

## 2. REVIEW OF RELATED LITERATURE
Much of the research regarding novice programmers explores the difficulties they encountered while learning to program. Our focus here will be on the research that explores the challenges novices face while learning to program.

### 2.1 The Difficulties of Programming

Learning to program is not easy. A number of studies have been carried out on what makes programming difficult. Researchers found that the difficulty might be caused by a lack of a mental model [2, 21, 28, 23], misconception of programming constructs [7, 13, 15, 20, 29], lack of programming strategies [3, 7, 26, 29, 33] and/or lack or absence of debugging strategies [1, 14].

This literature informs our analysis of the behavior of novice programmers, but does not directly address our fundamental question of how to identify at-risk students and, later, how we can intervene to guide them towards greater comprehension and retention. Through this research undertaking, we hope to shed more light on the errors that novices commit and the compilation behaviors they exhibit that may be indicative of poor understanding of the subject matter. These findings may later help us prescribe interventions that may promote better learning.

## 2.2 Common Errors Encountered

A number of studies have that looked at novice programming errors. Hristova and her team (Hristova, et al., 2003) conducted a survey among teaching assistants, professors and students from different schools in the United States and the members of the Special Interest Group for Computer Science Education (SIGCSE). Results from faculty and students were combined and divided into three categories: syntax errors, semantic errors and logical errors. A total of 62 error types were reported and they identified the top 20 from the list. It is noted that syntax errors are prevalent making 65% of the top 20 errors.

The studies of Jackson et.al and Jadud both collected online protocols in their studies. Students were taught Java. Table 1 identifies the top ten errors in Jackson's study [9] and Table 2 identifies the top five errors in Jadud's study[10].

**Table 1. Top ten errors in Jackson's study**

| Rank | Error | Percentage |
|---|---|---|
| 1 | cannot resolve symbol | 14.6 |
| 2 | ; expected | 8.5 |
| 3 | illegal start of expression | 5.7 |
| 4 | class or interface expected | 4.6 |
| 5 | identifier expected | 4.5 |
| 6 | ) expected | 3.8 |
| 7 | incompatible types | 2.8 |
| 8 | Int | 2.5 |
| 9 | not a statement | 2.5 |
| 10 | } expected | 2.3 |
| | Total | 51.8% |

**Table 2. Top five errors in Jadud's study**

| Rank | Error | Percentage |
|---|---|---|
| 1 | missing semicolon | 18% |
| 2 | unknown symbol: variable | 12% |
| 3 | bracket expected | 12% |
| 4 | illegal start of expression | 9% |
| 5 | Unknown symbol: class | 7% |
| | Total | 58% |

A comparison of both table shows that students beginning to program in Java seems to encounter almost the same error types. Most of these errors are syntax errors. These result confirms what Soloway and Spohrer concluded in their study: A few types of error accounts for the majority of errors dealt with by students [29]. The top ten errors comprised more than 50% of all the errors encountered.

# 3. THEORETICAL FRAMEWORK

How do novice programmers confront compile-time errors? Do they search their code for errors? Do they seek help from their notes, their peers or their teachers? Do they adopt a trial-and-error method of experimentation? Or do they give up?

Perkins, et al. [21] classified students into three stylistic categories: Stoppers, Movers, and Extreme Movers. Movers are students who persisted in experimentation and testing. They try one solution after another, using feedback effectively to progress in their programming tasks. Eventually, Movers succeed in arriving at a solution.

At the opposite end of the spectrum are the Stoppers. Stoppers are students who are unable to proceed because they have simply given up. These tend to be students who are frustrated or else had negative experiences with programming. They are unsure of themselves and lack confidence in handling the programming language and are not confident about how to get the machine to do what they need it to do.

Extreme Movers are those who tend to ignore feedback or use it ineffectively. They make program changes at random and hence do not progress effectively in their programming tasks. They tend to move around in circles or else go from one unworkable course of action to another, instead of moving progressively towards a real solution.

These classifications of programmer behaviors are of interest to computer science educators because they afford us with the opportunity to diagnose learning or non-learning behaviors and possibly intervene in order to promote greater learning. We are especially concerned with at-risk novices who may well decide to disengage from information technology-related courses altogether, opting instead for something less technical in nature. The challenge is to formalize these classifications into computationally tractable models.

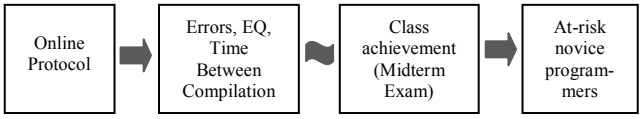
Having observed that students frequently encounter syntax errors, Jadud was able to quantify the compilation behavior of students in a given session. The type of syntax error that had been made and how often it had repeated is an indicator of how well or poorly a student was progressing. A penalty was assigned to behaviors that did not move a student towards the goal of having an error-free code.

Jadud developed a quantification of the student's compilation behavior through a grounded theoretic process. He called it the error quotient or EQ. Every record in the data logs represents one compilation event. Stored in each record is the error message if there was an error at the time of compilation, the location of the error in the file which is reported by the compiler as a line number, and the source code. To compute the EQ, we get the error message, the line number and the text of the source code. Given two compilation events, we compare if the error message is the same, are the line numbers the same, and the source code if the edit location is the same. Every pair of compilation event gets a normalized score and then averaged to get the final EQ score of the session.

The EQ characterizes how much or how little a student struggles with syntax errors while programming. An EQ score is of the range 0 to 1.0, where 0 is a perfect score. An EQ score of 0 does not mean that the student made no syntax errors in their programming process. What it means is that at no point did the

students encounter the same syntax error on two successive compilations of their program. Whereas a session scoring 1.0 means that every compilation resulted to the same syntax error all the time. Jadud's EQ will be used in scoring compilation behavior of novices in this study.

From the online protocols of the students, we shall extract the errors encountered, time between compilations and compute the EQ score. Correlation will be used to determine significant relationships among these three variables. Together with class achievement, we hope to identify at-risk students in an introductory programming class.




# 4. 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 participants of this study are the students enrolled in CS21 A-Introduction to Computing I, popularly called CS1 in the literature. A total of 143 students agreed to participate in the study, 17% were female and 83% were male. The DISCS have seven computing laboratories that also serve as lecture rooms. The CS 21A classes were held in three of these laboratories. The machines are connected in a local area network and the Internet. All the machines were installed with the same operating system, Java standard development kit and BlueJ.

Students in this study used BlueJ in performing their programming exercises. BlueJ is an integrated Java environment specifically designed for introductory teaching. The aim of BlueJ is to provide an easy-to-use teaching environment that facilitates the teaching of Java to students learning to program. To check for errors in a program, in the text editor students will simply click on the Compile button or press the keyboard shortcut for invoking the compiler. So as not to overwhelm beginners with error messages, BlueJ only shows one syntax error at a time. The line in question is highlighted, and the error message is reported at the bottom of the editor window.

Five laboratory exercises were designed to be performed by all the students on their scheduled laboratory day. Data was collected on this laboratory exercises. The exercises 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 is given in each exercise for the students to test their code.

## 4.1 Tools for Data Collection

The tool used in data collection was developed by Matthew Jadud and Poul Henriksen at the University of Kent. The data gathering tool is implemented as an extension to the BlueJ programming environment. BlueJ offers an extension API that allows third parties to develop extensions to the environment. Extensions offer additional functionality that are not included in the core system. In this study, the extension added to BlueJ sends data to the server whenever the user clicks the Compile button. Figure 1 shows a diagram of the data collection configuration.

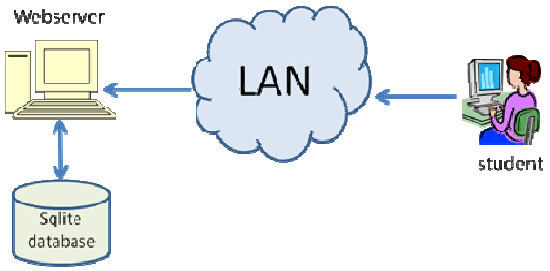



**Figure 1. Data Collection Configuration**

The focus of this study is on the compilation activities of the students inside the BlueJ environment. Each time the Compile button is clicked, BlueJ sent data to the server.

### *4.1.1 Databases*

BlueJ stores data in two tables in an Sqlite database[1]: `CompileData` and `InvocationData`. The `CompileData` table allows us to answer questions about student compilation behavior in the BlueJ programming environment. The `InvocationData` table stores data when a student invokes or executes a program. This study will focus on the `CompilationData` table only.

### *4.1.2 The `CompileData` Table*

A compilation event data is captured every time a student clicks the Compile button in the BlueJ IDE. Table 3 describes the fields of the `CompileData` table that will be extracted and used for data analysis.

**Table 3. Description of fields used in data analysis**

| Field | Description |
|---|---|
| `SESSION_ID` | a number generated by the data gathering program to identify a programming session |
| `DELTA_START_TIME` | the time the compile button is pressed |
| `DELTA_END_TIME` | the time the compiler finished compiling |
| `FILE_NAME` | the name of the file compiled |
| `FILE_CONTENTS` | the contents of the compiled file |
| `COMPILE_SUCCESSFUL` | Stores the value 1 if the compiler did not encounter an error, otherwise a 0 is stored, |
| `MSG_TYPE` | the type of message generated by the compiler: ERROR if an error was encountered, WARNING if a warning was encountered |
| `MSG_MESSAGE` | the error message generated by the compiler |
| `MSG_LINE_NUMBER` | the line number where the error was encountered, -1 if there was no error |

[1] SQLite is a small C library that implements a self-contained, embeddable, zero-configuration SQL database engine. SQlite is freely available at http://www.sqlite.org.

## 4.2 Procedure

On the first day of classes students were informed about the study. A consent letter was given to each of the students where they affixed their signature and the signature of their parent/guardian if they were willing to be part of the study. They were not obliged to join the study. They did not need any kind of preparation or follow any procedure during the study itself. All they had to do is attend their classes, complete the laboratory exercises, and behave as normally as possible.

Data gathering of the compilation logs was completely automated. Data was gathered only on the scheduled laboratory schedules when the standard exercises were given. The laboratory session lasted for an hour to one hour and a half. The lab exercises followed the lectures of the topics covered in the exercises. The data was collected on the first nine weeks of the First Semester of School Year 2007-2008.

## 4.3 Data Analysis

After all the data was gathered, the data was cleaned. Fields necessary for data analysis was extracted. The EQ per student was computed using these fields. The results of the EQ were then correlated with the students' midterm and final exam scores.

### *4.3.1 Data Cleaning*

When the server is turned on, all events inside BlueJ are captured. Sometimes students work on sample programs before working on their assigned laboratory exercises. All compilations and invocations on those sample programs were captured by the server.

Data recorded that were not part of the data to be collected were removed from the database. These data were removed manually using SQlite Administrator. Each database of each student per lab exercise needed to be opened and cleaned individually. The field `FILE_NAME` was used to identify records that were removed. In all the laboratory exercises, the students are told explicitly what will be the filename of the programs that they will create. A filename that is not required in the laboratory exercises was a candidate for removal.

### *4.3.2 Data Extraction*

Not all fields in the compile data table were used for data analysis. The following fields were extracted from the database for use during the data analysis: `SESSION_ID, DELTA_START_TIME, DELTA_END_TIME, FILE_NAME, FILE_CONTENTS,COMPILE_SUCCESSFUL, MSG_TYPE, MSG_MESSAGE, MSG_LINE_NUMBER.`

The extracted data were stored in a text file which was later used in generating summaries.

### *4.3.3 Generating Summaries*

To answer our research questions we generated the following summaries from the data extracted. In generating these summaries, we created Java programs that read the data from the text files, store the data to a corresponding Java class and generate the summaries.

1. A frequency count of compilation errors encountered
   a. Per class or section
   b. Over all laboratory exercises

The count of compilation errors encountered showed which programming constructs the students had difficulty using. From the individual count, we could identify which students were struggling with what type of errors. From the class count, we could quantify the progress of the class as a whole in terms of how good they were at resolving compilation errors. From the laboratory exercise count of all classes or sections, we could see which types of errors the participants were struggling with in relation to the exercise they were performing. Finally, the overall count of compilation errors covered all errors encountered in all laboratory exercises of all participants.

2. A frequency count of time between compilations
   a. Per Laboratory Exercise
   b. Overall laboratory exercises

Time between compilations was computed by subtracting the DELTA_START_TIME of every record in the database with the same SESSION_ID. The time computed was stored in ten second bins, to produce the number of compilations in every ten second interval.

3. Computation of the error quotient of each student
   a. Per laboratory exercise
   b. Over all laboratory exercises

The following is the algorithm for calculating the error quotient of a session, illustrated in Figure 2. Given a session of compilation events $e_1$ through $e_n$:

1. **Collate** Create consecutive pairs from the events in the session, for example, $(e_1,e_2),(e_2,e_3)...(e_{n-1},e_n)$.
2. **Calculate** Score each pair according to the algorithm presented in Figure 2.
3. **Normalize** Divide the score assigned to each pair by 9 (the maximum value possible for each pair).
4. **Average** Sum the scores and divide by the number of pairs. This average is taken as the error quotient (EQ) for the session.

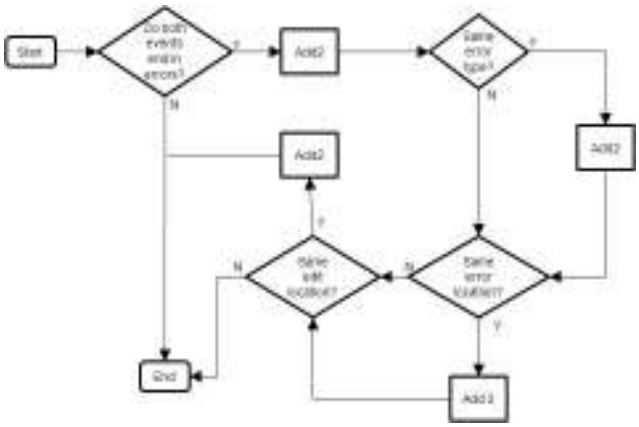



**Figure 2. To calculate the error quotient of a session, each pair of events is first scored using this algorithm. Those values are then summed and normalized, assuming a maximum score of 9 per pair.**

### *4.3.4 Determining Significance of Results*

After getting the errors encountered, time between compilation and EQ scores, we used Pearson's correlation to determine whether:

- there is a relationship between student's EQ score and achievement in class

The student's achievement in midterm exam and final exam is used in correlating with their EQ score.

## 5. RESULTS

The tools that we use for data collection allowed us to capture a copy of the student's work in progress every time they compiled their code. In the five laboratory sessions that we gathered data, a total of 28,386 compilation events were collected.

## 5.1 Errors Encountered

Of the 28, 385 compilation events that we collected,59% or a total of 16, 631 compilation events generated an error. Table 4 lists the top ten errors encountered and their percentage.

**Table 4. Top Ten Errors Encountered**

| Error Type | Percentage |
|---|---|
| 1. cannot find symbol - variable | 20% |
| 2. ';' expected | 13% |
| 3. ')' or ')' or'[' or ']' or '{' or '}' expected | 10% |
| 4. missing return statement | 8% |
| 5. cannot find symbol - method | 6% |
| 6. illegal start of expression | 6% |
| 7. incompatible types | 4% |
| 8. <identifier> expected | 4% |
| 9. class, interface, or enum expected | 3% |
| 10. 'else' without 'if' | 2% |
| Total | 76% |

There were a total of 52 different error types encountered. It is noted that the top ten errors account for 76% of all these errors. This means that majority of the students' time is spent correcting only a few different error types . Comparing this result to that of Jackson [9] and Jadud's [10] study shows that our subjects are encountering similar errors in their programming.

## 5.2 Time Between Compilations

The time between compilations gave us an idea of how much time students spent correcting or editing their code. Each bar in the histogram of Figure 3 shows time between compilations in every ten seconds window; 55% of all compilation events occurred in less than 30 seconds after the previous event. We question whether rapid-fire compilation is a quantitative description of Perkins, et al.'s [21] definition of Extreme Movers, and therefore indicative of a non-learning behavior. We can also see that, 15% of the time, students spend up to two minutes working on their code between compilations. We again question whether this time lag is a quantification of Movers' behavior and whether it implies careful examination and reflection upon code. These are questions that we will explore in future work.

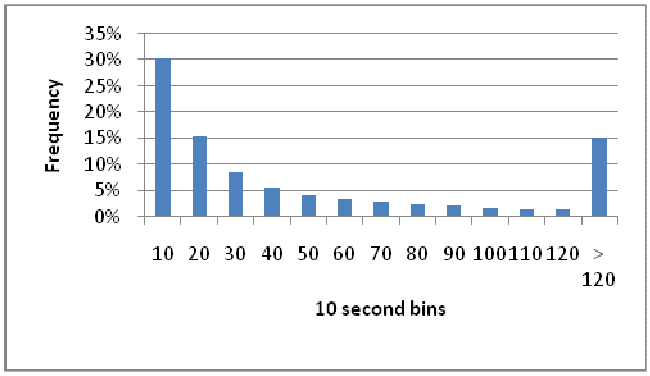



**Figure 3. Time between compilations, 10 second bins**

## 5.3 Error Quotient (EQ)

The error quotient is a measure of how many syntax errors a student encountered during a laboratory session. Students who encountered many syntax errors and failed to fix them from one compilation to the next, end up getting high error quotient. It gets even higher if an error persists successively in every compilation. On the other hand, students who have few syntax errors, or who correct their syntax errors quickly will end up with low error quotients. In other words, the higher the EQ score the more the student struggled with syntax errors and the longer it took them to correct the errors. The lower the EQ score, the better the student at correcting syntax errors.

We took the mean EQ score of the participants in five lab seesions: the lowest EQ score is 0.10 and the highest EQ score is 0.46. Figure 4 shows the percentage of the number of students getting an EQ score at intervals of 0.1: 23% got at an EQ score in the range >0.1-0.2, 38% in the range >0.2-0.3, 36% in the range >0.3-0.4, and 3% in the range >0.4-0.5.

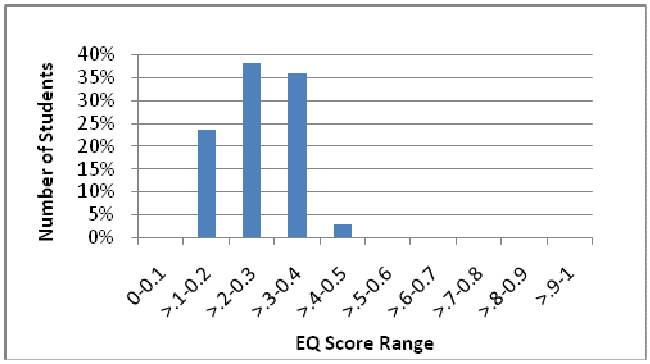



**Figure 4. Distribution of EQ Scores**

We then correlated the mean EQ score with the midterm exam scores and arrived at a correlation value R = -0.54 with p-value < 0.001. This is a significant result, implying that the lower the EQ score, the higher the midterm exam score of the student. The value of R means we have a moderate correlation between the EQ score and the midterm exam score. What this suggests is that the errors our students are encountering are not all intentional. Some of the errors encountered were not caused by students' misunderstanding. Some of these errors could be due to

carelessness. Even expert programmers can sometimes forget to type a semicolon at the end of a statement.

## 6. DISCUSSION

As the error analyses showed, only a small number of errors accounted for the majority of errors encountered. Most of the error types were syntactic in nature. These were simple errors which may be due to the fact that students were not yet used to writing programs and were not yet very familiar with the syntax. First on the list is *cannot find symbol-variable*. There are several possible explanations for this:: first, Java is case sensitive. The student may have capitalized a variable or a part of a variable while coding. Second, students tended to forget to declare variables first before using them.

The second most common error was *missing semicolon*. Beginning programmers often forget to put semicolon at the end of a statement. However, this error could mean something else. A semicolon may not have been expected at all. This may have been caused by a missing opening brace.

The third most common error is *forgetting to pair a parenthesis, or bracket or a brace*. A missing parenthesis though may not literally mean that a parenthesis is missing. It can be a missing argument or improper use of a method.

Though it may seem that the top three errors are simple, Jadud [11, 12] found that a missing semicolon can take to as much as thirty minutes or more for a novice to correct. The studies of [10] and [14] show that some students waste a considerable amount of time correcting syntax errors. This must be addressed because students can get disheartened getting errors every time they compile which may lead to frustration in programming. One possible way of improving the debugging ability of novices is to inform them of the common errors encountered by beginning programming students and discuss to them when these errors occur and how to solve them.

With regards to compiling behavior, it was surprising to see how quickly students compile their programs. It seemed that students were not giving much thought on their code between compilations. More than 50% of the compilations happened in less than 30 seconds of the previous compilation. This implied that students engaged in a certain amount of trial-and-error. Whether this is indeed the case is the subject of our future work. Among other things, we will need to note what type of error occurred and whether it was corrected or whether it persisted.

Interestingly enough, the tail of the histogram shows that 15% of the compilations happened two minutes after the previous compilation. A closer examination of this phenomenon is also part of future work. Among other things, we will need to note whether there was a large or small edit distance between the complied files. We will also correlate time between compilations and grades. Perhaps students with shorter compilation cycles have poorer grades knowing that they do not give much thought to the code before compiling it again.

Finally, the correlation of the EQ score and midterm exam score was significant based on its p-value. This meant that students with high EQ struggled with syntax while learning to program. Although this may seem an obvious result, it should be noted that the context of this study is to use EQ as a real-time diagnostic. This finding suggests that it might be possible to compute the EQs of students during their lab work and then use it as a basis for identifying at-risk students even before the midterms.

## 7. CONCLUSION

Most novices find programming difficult. In this study we attempted to understand students programming difficulties by capturing and analyzing our students' online protocols while performing their laboratory exercises. From their online protocols we derived the errors they encountered, computed for the time between compilations, and computed their Error Quotient (EQ),.

So far, we have learned what are the errors commonly encountered by our students and how often they compile their programs. Further analysis will be done to know how long it took the students to correct these errors and correlate with their exam scores. We want to find out if there is a relationship between the errors encountered and the student's achievement in class. We also want to find out if there is a pattern of error-types that identifies at-risk novice Java programmers. And lastly, we want to know if there is a relationship between time between compilations and the students' achievement in class.

We found that students who have high EQ scores tends to get lower midterm exam scores. In other words, students who are getting high EQ scores are those who may need some help. They could be struggling and may be at-risk of failing in programming.

The findings from this study leads to other questions for further examination: is there a correlation between time between compilations and student achievement? Do low-performing students and high-performing students encounter different sets of compile-time errors? Based on EQ, errors, and time between compilations, is it possible to establish a computationally tractable profile of Stoppers, Movers, and Extreme Movers? Finally, can these models be used to help identify and assist at-risk students early in the semester? By spotting at-risk students early and providing them needed intervention and help, CS Educators may reduce the drop-out rates of their programming classes.

## 8. ACKNOWLEDGEMENT

The authors thank Christine Amarra, Andrei Coronel, Jose Alfredo de Vera, Sheryl Lim, Ramon Francisco Mejia, Jessica Sugay, Dr. John Paul Vergara and the technical and secretarial staff of the Ateneo de Manila's Department of Information Systems and Computer Science for their assistance with this project. We thank the Ateneo de Manila's CS 21 A students, school year 2006-2007, for their participation. Finally, we thank the Department of Science and Technology's Philippine Council for Advanced Science and Technology Research and Development for making this study possible by providing the grant entitled "Modeling Novice Programmer Behaviors Through the Analysis of Logged Online Protocols."

## 9. REFERENCES

[1] Ahmadzadeh M., Elliman D. and Higgins C. ,"An analysis of patterns of debugging among novice computer science students", *Proceedings of the 10th annual SIGCSE conference on Innovation and technology in computer*

*science education*. - New York, NY, USA , ACM Press, pp. 84--88. 2005.

[2] Ben-Ari M. "Constructivism in computer science education," *ACM Press* New York, NY, USA, Vol. 30. 1998.

[3] Byckling P. and Sajaniemi J. ,"Roles of variables and programming skills improvement," *Proceedings of the 37th SIGCSE technical symposium on Computer science education,* ACM Press New York, NY, USA, pp. 413--417, 2006.

[4] Chmiel R. and Loui M.C., "Debugging: from novice to expert", *Proceedings of the 35th SIGCSE technical symposium on Computer science education.* - NY, USA : ACM Press New York, pp. 17--21, 2004.

[5] du Boulay B. ,"Some difficulties of learning to program", *Journal of Educational Computing Research*, Vol. 2. pp. 57--73, 1986.

[6] Ducasse M. and Emde A.M. ,"A review of automated debugging systems: knowledge, strategies and techniques", *Proceedings of the 10th international conference on Software engineering.* - Los Alamitos, CA, USA : IEEE Computer Society Press, pp. 162--171, 1988.

[7] Garner S., Haden P. and Robins A. ,"My program is correct but it doesn't run: a preliminary investigation of novice programmers' problems", *Proceedings of the 7th Australian conference on Computing education.* - Darlinghurst, Australia : Australian Computer Society, Inc., Vol. 42. - pp. 173--180, 2005.

[8] Hristova M. [et al.] ,"Identifying and correcting Java programming errors for introductory computer science students", *Proceedings of the 34th SIGCSE technical symposium on Computer science education*. - New York, NY, USA : ACM Press, pp. 153--156, 2003.

[9] Jackson J., Cobb M. and Carver C. ,"Identifying top java errors for novice programmers ", *Proceedings of the 35thASEE/IEEE Frontiers in Education Conference.* - Indianapolis, USA : [s.n.], 2005.

[10] Jadud M.C. ,"A First Look at Novice Compilation Behaviour Using BlueJ",*Computer Science Education*. - [s.l.] : Taylor & Francis, - 1: Vol. 15. - pp. 25--40, 2005.

[11] Jadud M.C. ,"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, 2006.

[12] Katz I.R. and Anderson J.R. ,"Debugging: An Analysis of Bug-Location Strategies ", *Human-Computer Interaction*. - [s.l.] : Lawrence Earlbaum, - 4 : Vol. 3. - pp. 351--399, 1987.

[13] Kay J. [et al.] ,"Problem-Based Learning for Foundation Computer Science Courses", *Computer Science Education*. - [s.l.] : Taylor & Francis, - 2 : Vol. 10. - pp. 109--128, 2000.

[14] Kummerfeld S.K. and Kay J. ,"The neglected battle fields of syntax errors ", *Proceedings of the fifth Australasian conference on Computing education.* - Darlinghurst, Australia : Australian Computer Society, Inc., pp. 105--111, 2003.

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

[16] Lane H.C. and VanLehn K. ,"Coached Program Planning: Dialogue-Based Support for Novice Program Design ", *Proceedings of the SIGCSE 2003*. - New York, NY, USA : ACM Press, pp. 148--152, 2003.

[17] Lewandowski G. [et al.] ,"What novice programmers don't know ", *Proceedings of the 2005 international workshop on Computing education research*. - New York, NY, USA : ACM Press, pp. 1--12, 2005.

[18] McCracken M. [et al.] ,"A multi-national, multi-institutional study of assessment of programming skills of first-year CS students ", A*nnual Joint Conference Integrating Technology into Computer Science Education.* - New York, NY, USA : ACM Press, pp. 125--180, 2001.

[19] McKeown J. and Farrell T. ,"Why We Need to Develop Success in introductory programming courses", *CCSC--Central Plains Conference.* - Maryville, MO : [s.n.], 1999.

[20] Milne I. and Rowe G. ,"Difficulties in Learning and Teaching Programming—Views of Students and Tutors", *Education and Information Technologies*, - [s.l.] : Springer, - 1 : Vol. 7. - pp. 55--66, 2002.

[21] Perkins D.N., Schwartz S. and Simmons R. ,"Instructional strategies for the problems of novice programmers", T*eaching and Learning Computer Programming: Multiple Research Perspectives.* - Hillsdale, NJ : Lawrence Erlbaum Asociates, pp. 153--178, 1988.

[22] Perkins DN ,"Conditions of Learning in Novice Programmers", *Journal of Educational Computing Research.* - 1 : Vol. 2. - pp. p37--55, 1986.

[23] Ramalingam V., LaBelle D. and Wiedenbeck S. , "Self-efficacy and mental models in learning to program", *Proceedings of the 9th annual SIGCSE conference on Innovation and technology in computer science education*. - New York, NY, USA : ACM Press, pp. 171--175, 2004.

[24] Ratcliffe MB , "Improving the Teaching of Introductory Programming by Assisting the Strugglers ", *The 33rd ACM Technical Symposium on Computer Science Education*. - Cincinnati, USA : [s.n.], 2002.

[25] Robins A., Rountree J. and Rountree N. , "Learning and Teaching Programming: A Review and Discussion", *Computer Science Education*. - [s.l.] : Taylor & Francis, - 2 : Vol. 13. - pp. 137--172, 2003.

[26] Sajaniemi J. and Kuittinen M. , "An Experiment on Using Roles of Variables in Teaching Introductory

Programming", *Computer Science Education.* - [s.l.] : Taylor & Francis, - 1 : Vol. 15. - pp. 59--82, 2005.

[27] Sanders K. and et.al , "A multi-institutional, multinational study of programming concepts using card sort data ", *Expert Systems*. - [s.l.] : Blackwell Publishing, - 3 : Vol. 22. - pp. 121--128, 2005.

[28] Sleeman D. [et al.] ,"An Introductory Pascal Class: A Case Study of Students' Errors", *Teaching and Learning Computer Programming: Multiple Research Perspectives*. - Hillsdale, NJ : Lawrence Erlbaum Asociates, pp. 237--257, 1988.

[29] Spohrer J.C. and Soloway E. , "Novice mistakes: are the folk wisdoms correct?", *Communications of the ACM.* - New York, NY, USA : ACM Press, - 7 : Vol. 29. - pp. 624--632, 1986.

[30] Spohrer J.G. and Soloway E. "Analyzing the high frequency bugs in novice programs ", *Papers presented at the first workshop on empirical studies of programmers on Empirical studies of programmers*. - Norwood, NJ, USA : Ablex Publishing Corp., pp. 230--251, 1986.

[31] Truong N., Roe P. and Bancroft P. "Static analysis of students' Java programs ", *Proceedings of the sixth conference on Australian computing education*. - Darlinghurst, Australia : Australian Computer Society, Inc., - Vol. 30. - pp. 317--325, 2004.

[32] Vee M.H.N.C., Meyer B. and Mannock K.L. , "Understanding novice errors and error paths in Object-oriented programming through log analysis ", *Proceedings of the Workshop on Educational Data Mining at the 8th International Conference on Intelligent Tutoring Systems* (ITS 2006), pp. 13-20, 2006.

[33] Winslow L.E. , "Programming Pedagogy--A Psychological Overview ", *SIGCSE Bulletin*, pp. 17--22, 1996.