ANALYZING NOVICE PROGRAMMERS' RESPONSE TO COMPILER ERROR MESSAGES*

Jonathan P. Munson
Math and Computer Science Department
Manhattanville College
2900 Purchase St
Purchase, NY 10577
jonathan.munson@mville.edu

Elizabeth A. Schilling
Center for Public Health & Health Policy
UConn Health
195 Farmington Ave 2100
Farmington, CT 06030
eschilling@uchc.edu

ABSTRACT

Error messages represent a critical feedback mechanism provided to introductory programming students during their early attempts at programming. However, these messages are explanations of program-translation problems, rather than true feedback on mistakes in a students program. This mismatch presents problems for novices, while experts are able to make more appropriate use of compiler error messages. Programming activity logs generated by an instructional programming environment were analyzed for students' utilization of error messages in their edits prior to each compilation. Overall, students addressed the first error about half the time. Assignment grades were positively associated with the proportion of times that the first error message was addressed.

INTRODUCTION

When learning a compiled programming language such as Java, the error messages produced by the compiler constitute a critical – and in the early stages, primary-feedback mechanism experienced by the student. (Several efforts have been made to make this

feedback easier for students to process – see Related Work, Improving the Feedback Mechanism).

However, compiler error messages are fundamentally the output of a mechanism whose role is to translate a program written in a high-level language into a machine-executable code. The output is the compiler's explanation of where the translation broke down. It is not a report of what is wrong with the program. (Researchers have attempted to produce reports that do actually identify programming errors-see Related Work, Characterizing Novice Mistakes).

This mismatch between the feedback programmers need in order to correct their mistakes and the information in the translation problems reported by the compiler is problematic for novices. Experienced programmers know how to respond to compiler error messages. Beyond quickly recognizing the import of such messages as "undefined symbol", they understand that a program translation problem is only a manifestation of a programming mistake, not necessarily the mistake that they actually made. When they see a "; expected" message, they know that it may or may not indicate a missing semicolon. They also know that if that is not the first error reported it may not even be an error. Error messages must be read carefully, and yet not necessarily taken literally. This apparently is difficult to learn.

We are interested in how students learn to develop mental models of the way programs work. For text-based, professional-oriented programming languages such as Java, learning the syntax through the edit→compile→interpret-errors cycle is an early step in that development. In this paper we report on our investigation of one particular aspect of making use of compiler error messages.

Fix the First Error First

Experienced programmers know that, when dealing with syntax errors, it is wise to correct the first error in the list produced by the compiler and then recompile. "*ALWAYS ALWAYS ALWAYS* ***fix the first compiler error first***" is the advice given by one instructor.[1] He goes on: "The first error is almost always a real error. Subsequent errors can simply be the result of the compiler being confused by the first error." This is also what we teach in our introductory programming course, but, based on the one-on-one help we gave during labs and office hours, it seemed a difficult lesson for students to learn. We are interested to know whether, overall, students in an introductory programming course can develop the sophisticated understanding of the compiler that will enable them to make better use of its error messages.

Specifically, we sought to answer these questions:

Q1: How does the likelihood of a student responding to the first error change as the course proceeds? That is, will students learn to address first errors within one

[1] http://users.csc.calpoly.edu/~phatalsk/references/ howToFixCompilerErrors.html

course?

Q2: Are better-performing students more likely to address the first error?

Using the activity logs generated by an instructional programming environment (see "Methodology" section), we synthesized observations such as "did address first error" from compilation events and performed statistical analyses to determine associations. In this paper we report on these analyses and discuss our results.

RELATED WORK

The edit→compile→interpret-errors cycle of novice programmers is of great interest to the research community, but for different reasons. We focus on the work that relates most closely to ours.

Characterizing Novice Mistakes

Many studies of mistakes made by novice programmers aim to better understand the difficulties facing students learning to program, and use logs of compiler errors as a kind of instrument to measure this. Jadud [6] collected compilation events from students using the BlueJ environment and, among other findings, reported the distribution of compiler error messages found. Whereas Jadud regarded compiler-reported errors as proxies for actual mistakes in programming, McCall and Kölling [10] also reported on error frequencies, but introduced a methodology that identifies actual programming mistakes instead of simply the compiler error messages. Their methodology requires manual coding of compilation logs. Altadmri and Brown [1] likewise reported on error frequencies, but used a much larger data set, and likewise aimed to identify actual programming mistakes rather than just compiler errors. As a fully automated scheme, their classification scheme is somewhat different than that in [10].

Helminen *et al* [5] discuss a programming environment that records detailed data about students' programming activities as well as execution errors from the Python interpreter. The authors' objectives for collecting the data go beyond understanding errors and extend to understanding how the students use the environment itself; for example, what kind of testing strategies they employ. ClockIt [13] is a system with similar objectives but based on the BlueJ environment. Retina [11] also logs student programming activities and focuses on compilation and execution errors. It uses this information while the course is ongoing to alert instructors to struggling students and allow students to see how their error "stats" compare with the rest of the class.

Studying Response to Error Messages

The above studies treat compiler error messages as a kind of assessment tool. Logs of errors are studied for the purpose of learning about the programmers who made them. Other studies recognize that error messages are an important pedagogical tool (whether or not they were designed with pedagogy in mind) and hence study their effectiveness at helping students produce correct programs.

Marceau *et al* [8] measured the effectiveness of the error messages of an instructional development environment, DrRacket. Their methodology involved manual coding of activity logs collected from a version of DrRacket instrumented for this purpose. Each edit was coded according to its level of response to the error message. For example, "[FIX]" indicated that the edit fixed the problem, and "[UNR]" indicated that the edit was unrelated to the error message. This level of specificity allows instructors to tailor interventions according to which error messages occur most frequently, and at what stage in the course. Although their analysis concerns the performance of DrRacket's error messages, the authors believe their coding rubric would apply to other languages.

Marceau *et al* also studied students' responses to error messages through interviews conducted with students while they worked through errors [9]. They recommend various improvements to DrRacket's error messages, from simplifying the vocabulary to being less proscriptive about where to correct a problem.

Improving the Feedback Mechanism

Several attempts have been made to make the messages produced by compilers easier to process for students. Gauntlet [3] rewrites compiler messages to explain them and provide guidance. Coull [2] and Nienaltowski et al [12] studied students' responses to longer-form, more descriptive error messages but report differing results on the efficacy of the messages. Coull reported success whereas Nienaltowski *et al* found that longer messages did not seem to aid message comprehension.

Deriving Intentions

We believe that attempts at rewriting and explanation of compiler error messages have a fundamental obstacle to their success in that they are based on the output of a device that has a purpose wholly different than pedagogy. A more effective approach may be to attempt an automated form of Lane and VanLehn's intention-based scoring [7] in which errors are regarded as deviations from a plan. The authors' focus on what they refer to as the "composition problem" matches our focus on giving feedback to students struggling with syntactically incorrect programs.

METHODOLOGY

We collected data on programming activity of 46 students in two sections of an introductory computer programming course over ten weeks of the course. Grade levels ranged from freshman to senior. Students used a new instructional programming environment we developed for use in our introductory programming courses. The system, which we call "Codework," is designed to streamline the assignment workflow, offer a simplified development environment, and provide data to support empirical studies of interventions. Codework offers students a browser-based user interface in which they develop their code and receive compilation and execution results. File storage,

compilation, and execution are performed on a server. Codework offers instructors their own Web-based interface to review and execute student code.

Program-editing events are captured continuously from the JavaScript-based editor and sent to the server at intervals, piggybacked on file-save requests. At the server, they are stored to a database. Compilation events, triggered explicitly when the student clicks the "Compile" button, are likewise logged to the database. For each event we capture the program source, the list of compiler error messages, the file name, the assignment the student is working on, a timestamp, the course section the student is in, and an ID for the student that is separate from and independent of the college-assigned ID.

From each compilation event we computed a "did address first error" variable and a "did address any error" as follows:

- Performed a diff (Myer's algorithm) between the source from this compilation event and the source from the last compilation event.
- If the line number of the first error in the list of compiler error messages from the previous compile matched a line number of one of the differences, "did address first error" was set True; if not, it was set False.
- If the line number of any error in the list of compiler error messages from the previous compile matched a line number of one of the differences, "did address any error" was set True; if not, it was set False.

We also created a variable representing an editing session. Edits made less than 90 minutes apart were considered in the same session.

RESULTS

Forty-six students completed six graded programming homework assignments and two in-class timed quizzes in the Codework system, resulting in 5879 observations.

Data analyses utilized generalized estimating equations (GEEs) using the GENLIN procedure in SPSS Version 22. The GEE approach accounts for correlated data due to clustering in which compilation attempts were nested within student [4]. GEE models are appropriate when the data come from a clustered design, when the correlation from the clustering is a "nuisance" parameter (i.e., is important to make correct inferences but is not of interest in itself), and when the objective of the analysis is to determine the effect of the predictors on the population-averaged response. These three criteria are present in our data.

Addressing First Error

Students addressed first errors during 48.2% of the compile attempts (**R1**). This percentage did not differ by type of assignment (homework (48.3%) or quiz (47.8%)), $p > .7$). In the full sample, the percentage of compile attempts in which the student addressed the first error differed by student ($\chi^2(45,5879)=165.08$, $p < .001$). In addition,

statistically significant differences among students were found in the subsets of data for 7 of the 10 assignments. Binary logistic GEE models investigated the effect of assignment on the probability of addressing the first error. In the first model, which included all graded assignments and quizzes, addressing the first error was predicted by assignment. The probability of addressing the first error differed by assignment ($\chi^2(7,4536)=21.08$, $p < .05$). Inspection of the coefficients revealed that the probability of addressing the first error decreased over the first 3 assignments and then increased for the next 2 and decreasing again for the last (**R2**). In a second model, the quizzes were contrasted with homework assignments. The grade the student received on the assignment was significantly associated with the probability of addressing the first error ($b=.333$, $p < .03$) (**R3**). In a model that predicted addressing the first error for quiz compile attempts by grade and time to compile, the probability of addressing the first error increased as the time to compile decreased (**R4**).

Time to Compile

In order to weed out outliers and restrict Times to likely single work sessions, only times less than 90 minutes were included in the models. The average time between compile attempts was (for times under 90 minutes) 2.4 minutes (1.57 minutes for quizzes and 2.67 minutes for homework assignments). The time in minutes between compile attempts was predicted using GEE linear models. In the first model, time between compile attempts was predicted by 1) whether the student addressed the first error, 2) the assignment, and 3) the grade. The assignment ($\chi^2(7,4536)=101.68$, $p < .001$) and grade ($\chi^2(1,4536)=4.59$, $p < .04$) significantly predicted Time to Compile, with the higher grade predicting longer time (**R5**). A second model comparing quizzes to homework assignments revealed that the time between compiling attempts was 1.1 minutes shorter for quizzes compared to homework assignments. In addition, a model predicting quiz compile attempts by whether or not the student addressed the first error determined that the time between compilations was .4 minutes on average (24 seconds) shorter for students who addressed the first error ($b= -.398$, $p < .01$) (**R6**).

DISCUSSION

We discuss the findings annotated R1-R6 above.

R1: *Students addressed the first compiler error about half the time.* Further analysis will be needed to determine what they are addressing the other 52% of the time. It is difficult to draw conclusions from this. It is possible that the practice of fixing only the first error was not well taught (by the first author). It is also possible that this practice is difficult to learn. There are simple interventions that could be implemented, such as automatically highlighting in the source the location of the first error.

R2: *The probability of addressing the first error decreased over the first three assignments and did not consistently increase after that.* We can understand a decrease in the percentage of time the first error was addressed, as students discover that it is not

easy to know what action to take in response to a compiler error message and therefore cast around blindly. But we had expected that following this we would see a consistent increase in the percentage of time the first error is addressed. However, our results do not allow us to confirm that. We gathered the data about two-thirds of the way through the course, so we may yet see that result when all the data is in. It may also be the case that a sophisticated understanding of the nature of compiler error messages does not develop until later.

The answer to Q1, then, is "don't know yet."

R3: *Higher assignment scores are positively associated with higher probability of addressing the first error.* Students who perform better are more likely to address the first error in the edit phase of the edit compile interpret-errors cycle. This result does not imply that addressing the first error resulted in the higher assignment scores. We find it more likely that the higher-performing students are more likely to learn the practice of addressing the first error first. We will need to refine our models before we can determine that having learned to address the first error enabled students to improve their assignment and quiz scores.

The answer to Q2, then, is yes.

R4, R6: *Behavior is different for timed quizzes than for assignments.* Student behavior in the edit compile interpret-errors cycle is different when under time pressure. When they address the first error they are more likely to recompile quickly.

R5: *A student's score on an assignment significantly predicts the time between their compilation events.* Jadud [6] observed that when students have just encountered a syntax error, they recompiled quickly, but when they have just compiled their code successfully, they are likely to spend a longer time editing before the next compile. Our finding that higher assignment scores are associated with longer times between compiles, together with Jadud's finding, suggests that higher-performing students are able to spend a longer amount of time in the edit phase of the edit→compile→interpret-errors cycle. A student who struggles with syntax more is likely to compile more frequently. Our findings that higher assignment scores are associated with longer times between compile attempts confirms, with statistical validity, the trends that Murphy *et al* observed with the Retina system [11]. This finding can be put to good use. In programming environments where time between compiles data is available to instructors in a timely way, alerts can be sent to the instructor about students that may be struggling. In environments where the data is available in real time, direct interventions with the student during their work session may be possible.

Limitations

One limitation in our study is that in this first dataset gathered, we did not explicitly demark sessions. Thus we were forced to synthesize it based on some uncertain assumptions. We will modify Codework to supply this marker for future semesters.

ACKNOWLEDGMENTS

Funding from the Manhattanville College Math and Computer Science Department paid for the cloud services that Codework runs on. This support is gratefully acknowledged.

REFERENCES

[1] Altadmri, A. and Brown, N.C.C. 37 Million Compilations?: Investigating Novice Programming Mistakes in Large-Scale Student Data. 522-527.

[2] Coull, N.J. 2008. *SNOOPIE: development of a learning support tool for novice programmers within a conceptual framework*. University of St Andrews.

[3] Flowers, T., Carver, C.A. and Jackson, J. 2004. Empowering students and building confidence in novice programmers through Gauntlet. *34th Annual Frontiers in Education, 2004. FIE 2004*. (2004).

[4] Hanley, J.A. 2003. Statistical Analysis of Correlated Data Using Generalized Estimating Equations: An Orientation. *American Journal of Epidemiology*. 157, 4 (2003), 364-375.

[5] Helminen, J., Ihantola, P. and Karavirta, V. 2013. Recording and Analyzing In-Browser Programming Sessions. *Koli Calling '13*. (2013), 13-22.

[6] Jadud, M.C. 2005. A First Look at Novice Compilation Behaviour Using BlueJ. *Computer Science Education*. 15, 1 (2005), 25-40.

[7] Lane, H.C. and VanLehn, K. 2005. Intention-Based Scoring: An Approach to Measuring Success at Solving the Composition Problem. *SIGCSE '05* (2005), 373-377.

[8] Marceau, G., Fisler, K. and Krishnamurthi, S. 2011. Measuring the Effectiveness of Error Messages Designed for Novice Programmers. *Proc SIGCSE*. (2011), 499-504.

[9] Marceau, G., Fisler, K. and Krishnamurthi, S. 2011. Mind your language: On novices' interactions with error messages. *Proceedings of the 10th SIGPLAN Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software - ONWARD '11. (2011)*, 3-17.

[10] McCall, D. and Kölling, M. 2014. Meaningful Categorisation of Novice Programmer Errors. *Frontiers In Education Conference* (2014), 2589-2596.

[11] Murphy, C., Kaiser, G., Loveland, K. and Hasan, S. 2009. Retina: helping students and instructors based on observed programming activities. *ACM SIGCSE Bulletin*. (2009), 178-182.

[12] Nienaltowski, M.-H., Pedroni, M. and Meyer, B. 2008. Compiler error messages. ACM SIGCSE Bulletin. 40, 1 (2008), 168.

[13] Norris, C., Barry, F. and Jr, J.F. 2008. ClockIt: collecting quantitative data on how beginning software developers really work. *ACM SIGCSE Bulletin* …. 40, 3 (2008), 37-41.