Coarse-Grained Detection of Student Frustration in an Introductory Programming Course

Ma. Mercedes T. RODRIGO
Dept. of Information Systems and Computer Science,
Ateneo de Manila Unversity,
Loyola Heights, Quezon City, Philippines
+63 (2) 426-6071
mrodrigo@ateneo.edu

Ryan Shaun J. d. BAKER
Human-Computer Interaction Institute
Carnegie Mellon University
Pittsburgh, PA
+1 (412) 268-9690
rsbaker@cs.cmu.edu

ABSTRACT

We attempt to automatically detect student frustration, at a coarse-grained level, using measures distilled from student behavior within a learning environment for introductory programming. We find that each student's average level of frustration across five lab exercises can be detected based on the number of pairs of consecutive compilations with the same edit location, the number of pairs of consecutive compilations with the same error, the average time between compilations and the total number of errors. Attempts to detect frustration at a finer grain-size, identifying individual students' fluctuations in frustration between labs, were less successful. These results indicate that it is possible to detect frustration at a coarse-grained level, solely from coarse-grained data about students' behavior within a learning environment.

Categories and Subject Descriptors

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

General Terms

Human Factors

Keywords

Novice Programmers, Affect, Frustration Detection

1. FRUSTRATION AND THE NOVICE PROGRAMMER

With the general decline in computer science and information technology courses enrollment over recent years [28] (despite some uptick in the current recession), computer science educators are growing increasingly interested in promoting student retention using a variety of approaches. Computer science research groups within universities recruit undergraduates into research projects early in the students' college lives [cf. 10, 26], games are used as capstone projects and/or to teach introductory computer science concepts [3], curriculum pace is reduced to give students more time to digest content [29], students program in pairs [30] or groups [13], and are guided to participate in peer-review [2] to stimulate conversations about their work.

A recurring theme in these efforts is the mitigation of frustration among novice programmers [cf. 10, 13, 29]. Students' self-perception of their programming competence [4, 20] and their comfort levels with the subject [4] have been shown to be predictors of programming achievement. Many students experience programming as a challenge to their self-esteem. Given the frustrating experience of encountering bugs and being unable to correct them immediately [23], a student may opt to disengage from the programming task.

Researchers recognize frustration is potentially a mediator for student disengagement and eventually attrition. They endeavor to detect frustration in order to intervene in ways that will help students persevere [14]. Models of frustration inform tutors (whether human or automated) so that these interventions maximize student self-efficacy and support effective learning [21].

Prior studies on the automated detection of student frustration have often relied upon facial expressions, eye-gaze, posture, and physiological signals such as skin conductance and blood volume pulse [cf. 9, 11, 14, 21]. The models from these experiments have been robust. D'Mello and his colleagues [9] were able to distinguish each student's affective state (out of a set of 5 affective states) 42% of the time, and to distinguish frustration from the neutral state 78% of the time. Kapoor et al's [14] model could accurately identify frustration 79% of the time. Finally, McQuiggan's model [21] reported an accuracy rate of 68%.

Thus far, detectors of affect have not been developed for the domain of computer programming. Khan [15] proposed that it may be possible to detect programmer moods based on keystroke and mouse movement level data, however no further work has been published to this end. One interesting challenge in detecting student affect in introductory programming courses comes from the differences between the types of learning environments typically used in these courses, as compared to more traditional intelligent tutoring systems. While intelligent tutors for programming exist [cf. 7, 22] students must eventually learn to program in free-form programming environments, which allow the student to edit entire programs to solve a problem. Within these environments, available data is either much finer-grained (mouse movements and keystrokes) or much coarser-grained (compilation by compilation) than the data collected in [9. 11. 14.

ICER'09, August 10–11, 2009, Berkeley, California, USA.

21]. A number of studies have examined student online protocols, defined as all student submissions to a compiler, to determine student misconceptions [cf. 19] but our literature did not uncover any studies that have used online protocols to study student affect.

We attempt to detect student frustration automatically based on measures distilled from student behavior within the program development environment. Automatic detection can support instructors in large introductory classes, where instructors do not have the time to monitor every student's affective experience.

2. METHODS

This study was conducted with Computer Science freshmen and Management Information Systems sophomores of the Ateneo de Manila University during the first semester of school year 2007-2008. The students were taking their first collegiate programming course, CS21A Introduction to Computing, called CS1 in the Computer Science Education literature. There were five sections of CS21A during this semester, with a total of 146 students. Although the teachers for each section varied, the textbook, presentation slides, examples, exercises, midterm exam, final exam, and programming projects were uniform. The programming language used in the course was Java. During the first half of the semester, the students used the BlueJ Integrated Development Environment (IDE) [17] to complete programming exercises and assignments. Ten students were randomly selected from each section for behavior and affect observation, for a total of 50 students. Students whose observation or compilation records were incomplete because of absences or technical problems were deleted from the dataset. After all the deletions, our sample was reduced to 40 students. Twenty-seven were male and 13 were female.

The CS21A classes individually completed the lab exercises within a computer laboratory. The lab sessions were 50 minutes long. They were part of regularly scheduled class time and were graded, so all students were expected to be present. Each student was assigned a permanent seat and computer for the semester. Over the first nine weeks of the semester, the students were asked to write five small programs. During these lab periods, students were free to consult their books, notes, presentation slides, classmates and the teacher.

As the students completed these programming exercises in BlueJ, the BlueJ IDE sent data about each student compilation to a SQLite database running in the background [18]. The saved data included but were not limited to the computer number, time stamp of the compilation, success or failure of the compilation, error message (if any), error message line number, name of the compiled file, and source code.

During each lab period, two trained observers noted each student's affective state and behavior. In each session, the observers were drawn from a pool of five students currently taking their master's degrees in either Computer Science or Education. Each of these observers had teaching experience.

The observers were synchronized by a timed PowerPoint presentation with slides numbered 1 to 150, each slide lasting 20 seconds. During each time slide, the observers surreptitiously looked at a student's facial expressions, body language, utterances, and interactions with the computer, fellow students or teacher. The observers studied the same 10 students per section, per lab period. The identities of the 10 students under observation were not revealed at any time. Observers wandered around the classroom and watched the subjects from a distance. Since the entire class occupied the lab during the lab period, it was fairly easy to disguise who exactly the observers were watching and at what time.

The observers then coded one affective state and one behavior for that student for that time period. The affective states coded were taken from [9, 25]. Key examples of behavior of students experiencing each affective state are given below:

1. **Boredom** –slouching, and resting the chin on his/her palm; statements such "This is boring!"
2. **Confusion –**scratching his/her head, repeatedly looking at the same interface elements; consulting with a classmate or a teacher; flipping through lecture slides or notes; statements such as "Why didn't it work?"
3. **Delight** –clapping hands, thrusting fists into the air as a sign of victory, laughing with pleasure; statements such as "Yes!" or "I got it!"
4. **Surprise** –jerking back suddenly or gasping; statements such as "Huh?" or "Oh, no!"
5. **Frustration –**banging on the keyboard or the mouse; cursing; statements such as "What's going on?!?"
6. **Flow** – complete immersion and focus upon the system [8]; towards the computer or mouthing solutions to him/herself while solving a problem
7. The **Neutral** state, which was coded when the student did not appear to be displaying any of the affective states above, or the student's affect could not be determined for certain.

The behavioral states coded were taken from [1, 25] and described as follows:

1. **On-task** – working on the programming task
2. **On-task conversation: Domain and problem-focused**– helping or asking for help from the teacher or another student about the program specifications or a Java construct
3. **IDE-related conversation** – helping or asking for help from the teacher or another student about the IDE
4. **Off-task conversation** – talking about any other topic
5. **Off-task solitary behavior** – behavior that did not involve the programming task or another person (such as surfing the web, blogging or checking a cell phone)
6. **Inactivity –** the student stares into space or puts his/her head down on the desk.
7. **Gaming the System** – sustained and/or systematic guessing, such as rapid-fire compiling without consulting the error messages; repeatedly requesting for help in order to arrive at a solution [cf. 1].

For tractability, observers coded only the first observed affective state and behavior per student per time slice. After 20 seconds, the observers moved on to the next student. After the 10^{th} student, the observers returned to the first. The observers gathered 15 affect and 15 behavior observations per student per lab period.

After observations for all five lab sessions were gathered, inter-rater reliability was computed. Cohen's [6] kappa was acceptably high at $\kappa=0.65$ for affect and $\kappa=0.75$ for behavior.

Recall that the purpose of this study is to detect frustration in an aggregate fashion across all labs and not to detect frustration as it happens, action-by-action. To this end, we constructed a per-student record with the student's frustration profile and a compilation profile per lab and across all five labs. To arrive at the frustration profile of each student, we first computed the percentage of observations in which each student exhibited frustration within each lab period. To arrive at the student's average frustration level, we took the mean of the student's frustration percentages across the five labs.

In the process of analyzing student online protocols, Jadud [18] defined a construct, the Error Quotient (EQ), that assessed how well or how poorly a student copes with syntax errors. The EQ was calculated based on number of pairs of consecutive compilations in error, number of pairs of consecutive compilations with the same error, number of pairs of consecutive compilations with the same error location, and number of pairs of consecutive compilations with the same edit location. Tabanao et al [27] showed that EQ correlated with mid-term test scores. To arrive at the compilation profile, we computed the number of pairs of compilations in error, number of pairs of compilations with the same error, number of pairs of consecutive compilations with the same edit location, and number of pairs of consecutive compilations with the same edit location (essentially the components of the EQ measure). We also computed for the average time between compilations, total number of compilations, and total number of errors. The student's overall compilation profile was the average of each of these measures across all five labs.

3. RESULTS AND DISCUSSION

We used Weka [31] to generate linear regression models of students' per-lab and average frustration.

Table 1 shows that a student's average frustration level for the five labs is equal to 0.0033 * number of consecutive compilations with the same edit location – 0.0036 * the number of consecutive pairs with the same error + 0.0002 * the average time between compilations + 0.0005 * the total number of errors -0.0109. Factors whose coefficients were too small to make an impact on the model were removed. This model achieves a weak *r* of 0.3178 when tested using 10-fold cross validation. Despite the weak correlation, the model is still likely to be accurate enough to drive some forms of "fail-soft" intervention, where the costs of a mis-assigned intervention are relatively low.

Table 1. Weka linear regression model, predicting average frustration across all labs.

Frustration =	*r* **10-fold Cross Validation**	BiC'
0.0033 * Consecutive pairs of compilations with the same edit location + -0.0036 * Consecutive pairs of compilations with the same error + 0.0002 * Average time between compilations + 0.0005 * Total errors + -0.0109	0.3178	-7.86

The generalizability of a model can be tested using the Bayesian Information Criterion for Linear Regression (BiC'), a method frequently used to assess the tradeoff between model fit and the number of parameters (which can spuriously increase model fit). Values of BiC' under -6 signify that the model has significantly better fit than chance, given the number of model parameters [24]. The BiC' value for this model is -7.86, which means that the model explains statistically significantly more of the variance in frustration than could be expected by chance, given the number of model parameters.

Attempts to detect student frustration on a per lab basis, using this sort of coarse-grained data, were significantly less successful (Table 2). As with the model in Table 1, factors whose coefficients were too small to make an impact on the model were removed. Two of the labs, labs 3 and 4, had good cross-validated r but poor BiC'. One lab, lab 5, had significant BiC' but poor cross-validated *r*. This pattern of results suggests that there is some trend towards the models generalizing better than chance (as both are measures of generalizability), but that the generalizability is unstable.

These findings have two major implications. First, we find that it is possible to automatically detect student frustration based on coarse-grained data such as compilation logs. In particular, we can infer that a student is frustrated when, over successive lab periods, he or she tends to make compilation errors in the same spot, despite successive edits. However, detecting frustration with data this grain-size appears to require several sessions worth of data for high accuracy; a single session's data appears to be insufficient to build a robust model.

The model presented here is hence insufficiently fine-grained for interventions at the lab-session level, but is sufficiently fine-grained to drive meaningful instructional interventions – after 5 sessions, the model can detect which students are experiencing frustration, information that can be provided to instructors so that these students can receive interventions from course staff before frustration turns into non-retention in computer science. Alternatively, the student could receive a message from the system sympathizing with their frustration [16] and encouraging them to keep trying, or reminding them that effort matters more than innate ability [cf. 12]

4. CONCLUSION

Within this paper, we present an automated detector that detects student frustration using coarse-grained data taken from protocols of the interaction between students and a programming environment. Using Weka, we generated a linear regression detector of average student frustration across five lab exercises. The detector made its prediction based on students' average number of consecutive compilations with the same edit location, average number of consecutive pairs with the same error, the average time between compilations, and average number of errors. This model predicted frustration significantly better than would be expected by chance, given the number of parameters in the model.

Our attempts to detect frustration on a per-lab basis were less successful, with detector performance quite unstable when generalized to new data (or under simulated generalization, using the BiC' measure). This implies that it is possible to detect frustration using coarse-grained data but the detectors (or the

training algorithms) need a substantial amount of data to for the models to be reasonably accurate.

Table 2. Weka linear regression model, predicting frustration in individual labs.

Lab No	Frustration =	*R* 10-fold Cross Validation	BiC'
1	0.0015 * Consecutive pairs of compilations in error + -0.0029 * Consecutive pairs of compilations with the same error + -0.0002 * Average time between compilations + -0.0014 * Number of compilations + 0.0016 * Total errors + 0.0555	-0.0135	11.06
2	-0.0026 * Consecutive pairs of compilations in error + 0.0066 * Consecutive pairs of compilations with the same error location + 0.0127	0.0136	-1.45
3	-0.0016 * Consecutive pairs of compilations in error + 0.0015 * Total errors + 0.0127	-0.2352	-2.43
4	0.0003 * Total errors + 0.0057	-0.3472	0.88
5	0.0014 * Consec pairs of compilations with the same edit location + -0.0013 * Consecutive pairs of compilations with the same error location + 0.0003 * Number of compilations + -0.0047	-0.0331	-6.78

In future work, we will attempt to develop a finer-grained detector for frustration and other affective states, using keystroke and mouse movement data as well as the coarse-grained semantic data utilized here. Thus far, most detectors of student affect have been based on data at only one grain-size; there may be leverage from combining distinct types of data to detect affect in a co-training (e.g. multiple unrelated signals combined into one model) process [5]. Determining how coarse-grained affect detectors can be meaningfully used to drive interventions, towards improving affect, learning, and course retention, is another important area of future work.

5. ACKNOWLEDGMENTS

The authors thank Matthew C. Jadud for very helpful discussions and collaborative support. We also thank 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, Sheila Pascua, Jessica Sugay, Emily Tabanao, 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 2007-2008, for their participation. This publication was made possible through the Department of Science and Technology's Philippine Council for Advanced Science and Technology Research and Development grant 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,** the Engineering Research and Development for Technology program for the grant entitled **Multidimensional Analysis of User-Machine Interactions Towards the Development of Models of Affect**, the Pittsburgh Science of Learning Center, National Science Foundation award SBE-0354420, and Dr. Rodrigo's 2008-2009 Advanced Research and University Lecturing Fulbright Scholarship from the US Department of State, the Philippine American Educational Foundation and the Council for International Exchange of Scholars.

6. REFERENCES

[1] Baker, R. S., Corbett, A. T., Koedinger, K. R., K.R., and Wagner, A. Z. 2004. Off-task behavior in the Cognitive Tutor classroom: When students "Game The System". *ACM CHI 2004: Computer-Human Interaction*, 383-390, 2004.

[2] Barker, L. J., Garvin-Doxas, K. and Roberts E. 2005. What can computer science learn from a fine arts approach to teaching?. In *Proceedings of the 36th SIGCSE Technical Symposium on Computer Science Education* (St. Louis, Missouri, USA, February 23 - 27, 2005). SIGCSE '05. ACM, New York, NY, 421-425.

[3] Barnes, T., Richter, H., Powell, E., Chaffin, A. and Godwin, A.. 2007. Game2Learn: building CS1 learning games for retention. In *Proceedings of the 12th Annual SIGCSE Conference on innovation and Technology in Computer Science Education* (Dundee, Scotland, June 25 - 27, 2007). ITiCSE '07. ACM, New York, NY, 121-125.

[4] Bergin, S. and Reilly, R. 2005. Programming: factors that influence success. In *Proceedings of the 36th SIGCSE Technical Symposium on Computer Science Education* (St. Louis, Missouri, USA, February 23 - 27, 2005). SIGCSE '05. ACM, New York, NY, 411-415.

[5] Blum, A. and Mitchell, T.. Combining labeled and unlabeled data with co-training.1998. In *Proceedings of the 11th Annual Conference on Computational Learning Theory (COLT '98)*, 92-100.

[6] Cohen, J. 1960. A Coefficient of Agreement for Nominal Scales. *Educational and Psychological Measurement,* 20, 37-46.

[7] Corbett, A. T. and Anderson, J. R. 1992. The LISP intelligent tutoring system: Research in skill acquisition. In J. Larkin, R. Chabay, and C. Scheftic (Eds.), *Computer Assisted*

Instruction and Intelligent Tutoring Systems: Establishing Communication and Collaboration. Hillsdale, NJ: Erlbaum.

[8] Csikszentmihalyi, M.. 1990. *Flow: The Psychology of Optimal Experience*. New York: Harper and Row.

[9] D'Mello, S. K., Craig, S. D. , Witherspoon, A., McDaniel, B. and Graesser, A. 2008. Automatic detection of learner's affect from conversational cues. *User Modeling and User-Adapted Interaction*, *18*, 45-80.

[10] Dahlberg, T., Barnes, T., Rorrer, A., Powell, E. and Cairco, L. 2008. Improving retention and graduate recruitment through immersive research experiences for undergraduates. In *Proceedings of the 39th SIGCSE Technical Symposium on Computer Science Education* (Portland, OR, USA, March 12 - 15, 2008). SIGCSE '08. ACM, New York, NY, 466-470.

[11] Dragon, T., Arroyo, I., Woolf, B. P., Burleson, W., Kaliouby, R. e., and Eydgahi, H. 2008. Viewing student affect and learning through classroom observation and physical sensors. In B. P. Woolf, E. Aimeur, R. Nkambou and S. P. Lajoie (Eds), *Intelligent Tutoring Systems 2008*.

[12] Dweck, C.. 2000. *Self-theories: Their Role in Motivation, Personality and Development*. Philadelphia, PA: Psychology Press.

[13] Guo. J. 2008. Using group-based projects to improve retention of students in computer science major. *Journal of Computing in Small Colleges, 23*(6), 187-193, 2008

[14] Kapoor, A., Burleson, W., and Picard, R. W. 2007. Automatic prediction of frustration. *International Journal of Human-Computer Studies*, *65*, 724-736.

[15] Khan, I. A., Hierons, R. M., and Brinkman, W. P. 2007. Mood independent programming. In *Proceedings of the 14th European Conference on Cognitive Ergonomics: invent! Explore!* (London, United Kingdom, August 28 - 31, 2007). ECCE '07, vol. 250. ACM, New York, NY, 269-272.

[16] Klein, J., Moon, Y. and Picard, R.W. 2002. This computer responds to user frustration: Theory, design, results, and implications, *Interacting with Computers*, 14, 119-140.

[17] Kolling, M. and Rosenberg, J. 2004. BlueJ v. 2.1.3. Computer Software.

[18] Jadud, M. C. 2006. An Exploration of Novice Compilation Behavior in BlueJ. Doctoral thesis. University of Kent, 2006.

[19] Lane, H. C. and VanLehn, K. 2005. Intention-based scoring: an approach to measuring success at solving the composition problem. In *Proceedings of the 36th SIGCSE Technical Symposium on Computer Science Education* (St. Louis, Missouri, USA, February 23 - 27, 2005). SIGCSE '05. ACM, New York, NY, 373-377.

[20] McKinney, D. and Denton, L. F. 2004. Houston, we have a problem: there's a leak in the CS1 affective oxygen tank. In *Proceedings of the 35th SIGCSE Technical Symposium on Computer Science Education* (Norfolk, Virginia, USA, March 03 - 07, 2004). SIGCSE '04. ACM, New York, NY, 236-239.

[21] McQuiggan, S. W., Lee, S., and Lester, J. C. 2007. Early prediction of student frustration. In A. Paiva, R. Prada, and R. W. Picard (Eds.), *Affective Computing and Intelligent Interaction*, 698-709.

[22] Ng Cheong Vee, M.-H., Meyer, B., and Mannock, K. L. 2006. Understanding novice errors and error paths in Object-oriented programming through log analysis. Proceedings of the Workshop on Educational Data Mining at Intelligent Tutoring Systems 2006, 13-20.

[23] Perkins, D. N., Hancock, C., Hobbs, R., Martin, F. and Simmons, R. 1985. Conditions of Learning in Novice Programmers. Concept Paper. Educational Technology Center, Harvard Graduate School of Education..

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

[25] Rodrigo, M. M. T., Baker, R. S. J. d., Lagud, M. C. V., Lim, S. A. L., Macapanpan, A. F., Pascua, S. A. M. S., Santillano, J. Q., Sevilla, L. R. S., Sugay, J. O., Tep, S. and Viehland, N. J. B. 2007. Affect and usage choices in simulation problem-solving environments. In R. Luckin, K. R. Koedinger, J. Greer (Eds.), *13th International Conference on Artificial Intelligence in Education*, 145-152.

[26] Stephenson, P., Peckham, J. , Hervé, J., Hutt, R. and Encarnação, L. M. 2006. Increasing student retention in computer science through research programs for undergraduates. In *ACM SIGGRAPH 2006 Educators Program* (Boston, Massachusetts, July 30 - August 03, 2006). SIGGRAPH '06. ACM, New York, NY, 10.

[27] Tabanao, E., Rodrigo, M. M. T. and Jadud, M. 2008. Identifying at-risk novice programmers through the analysis of online protocols. Philippine Computing Society Congress 2008, (UP Diliman, Quezon City, February 23-24, 2008).

[28] Vegso, J. 2008. Enrollments and degree production at US CS departments drop further in 2006/2007. *CRA Bulletin*. Accessed 3 December 2008 from the CRA Bulletin web site: http://www.cra.org/wp/index.php?p=139,.

[29] Whittington, K. J. and Bills, D. P. 2004. Alternative pacing in an introductory java sequence. In *Proceedings of the 5th Conference on information Technology Education* (Salt Lake City, UT, USA, October 28 - 30, 2004). CITC5 '04. ACM, New York, NY, 118-121.

[30] Williams, L. and Upchurch, R. L. 2001. In support of student pair-programming. *ACM SIGCSE Bulletin, 33*(1), 327-331, 2001.

[31] Witten, I. H. and Frank, E. 2005. *Data Mining: Practical Machine Learning Tools and Techniques*, 2nd Edition, Morgan Kaufmann: San Francisco, 2005.