Applying Educational Data Mining to the Study of the Novice Programmer, within a Neo-Piagetian Theoretical Perspective

Alireza Ahadi

University of Technology, Sydney
Australia
Alireza.Ahadi @student.uts.edu.au

Keywords: *POP-I.B. barriers to programming, POP-II.A. novices, POP-V.A. Neo-Piagetian, POP-VI.E. computer science education research*

1. Introduction

It is well known that many first year undergraduate university students struggle with learning to program. Educational Data Mining (EDM) applies data mining, machine learning and statistics to information generated from educational settings. In this PhD project, I will apply EDM to study first-semester novice programmers, using data collected from students as they work on computers to complete their normal weekly laboratory exercises.

This PhD project is within its first six months.

2. Background

My doctoral project is informed by past research from two broad sources, described below.

2.1 Educational Data Mining and the Novice Programmer

Over the last ten years, a number of systems have been developed systems for routinely logging data about computing students (Winters & Payne, 2005; Jadud, 2006; Norris et al., 2008; Edwards et al., 2009; Edwards, 2013). Most of these systems are used primarily to grade student assignments, both summatively and formatively. Consequently, much of the data collected by these systems relates to frequency of submission and number of test cases passed. Other systems have been developed to collect richer data (Romero et al., 2004 & 2005; Brown et al., 2014).

2.2. Neo-Piagetian Theory

Neo-Piagetian provides the theoretical context for this project. Lister (2011) proposed a three stage model of the early stages of learning to program, which are (from least mature to most mature) the sensorimotor stage, the preoperational stage and the concrete operational stage. For the benefit of readers unfamiliar with neo-Piagetian theory, we refer to these three stages by different names from Lister (2011), and define each as follows:

1) **Pre-tracing Stage (i.e. sensorimotor):** The best known problems faced by novices at this level are misconceptions about the semantics of programming language constructs. For example, a novice might believe that assignment statements assign values from left to right. There also exist other problems, which result in the novice being unable to accurately trace (i.e. manually execute) small pieces of code (du Boulay, 1989). When attempting to write code, novices at this stage may struggle to produce code that compiles successfully. When they do produce a "clean compile" the code may bear little resemblance to the problem to be solved, and the code may even contain aspects that are bizarre to an experienced programmer.
2) **Tracing stage (i.e. preoperational):** The novice can reliably manually execute ("trace") multiple lines of code. Furthermore, novices at this stage rely almost exclusively on their tracing skill when reasoning about programs. When reading code, these novices often make inductive guesses about what the code does, by performing one or more traces, and examining the relationship between the input and resultant output. When writing code, these novices tend to patch and repatch their

code, on the basis of either (1) their results from tracing through their code specific initial variable values, that may be chosen somewhat at random chosen, or (2) after a tutor has pointed out a bug (Ginat, 2007). These novices struggle to truly design a solution.

3) **Post-tracing Stage (i.e. concrete operational):** These novice programmers reason about code deductively, by reading the code itself, rather than using the preoperational inductive approach. They can relate diagrams to code. This stage is the first stage where students begin to show a purposeful approach to writing code.

Since that first paper by Lister in 2011, he and his colleagues have collected empirical support for these neo-Piagetian stages (Corney, 2012; Teague et al., 2012a, 2012b, 2013; Teague and Lister, 2014a, 2014b, 2014c and 2014d). However, all these studies have involved students completing programming tasks with pencil and paper. An example of a task given to novices in these pencil and paper exercises is as follows:

> The purpose of the following Java code is to move all elements of the array x one place to the right, with the rightmost element being moved to the leftmost position:

```
int temp = x[x.length-1];

for (int i = x.length-2; i>=0; --i)
   x[i+1] = x[i];

x[0] = temp;
```

> Write code that undoes the effect of the above code. That is, write code to move all elements of the array x one place to the left, with the leftmost element being moved to the rightmost position.

Other examples of tasks used to identify the neo-Piagetian stage of a novice programmer can be found in this conference proceedings (Ahadi, Lister and Teague, 2014; Teague and Lister, 2014d; Teague, 2014e).

My research will be the first to look for evidence for these neo-Piagetian stages in novices as they write programs on a computer. One or both of two broad approaches will be used for detecting relationships between on-machine programming behaviour and neo-Piagetian stages:

- **Supervised Learning:** Volunteer students will first complete paper-based tests, while thinking out loud, as in Teague's research, using the existing tasks designed by Teague and Lister. These tests will be used to assign novices to a neo-Piagetian stage. Their subsequent on-machine programming behaviour will then be analysed by data mining algorithms to try to detect differences between students assigned to the different neo-Piagetian stages.
- **Unsupervised Learning:** Automatic cluster analysis will be applied to data collected from the on-machine programming behaviour of novices. Manual analysis will then attempt to link clusters to neo-Piagetian stages.

What neo-Piagetian related patterns might we find in the data, and what problems could there be in automatically detecting those patterns? With reference back to the neo-Piagetian stages described in section 2.2:

- **Pre-tracing Stage (i.e. sensorimotor):** *These novices struggle to produce code that compiles successfully. When they do produce a "clean compile" the code may bear little resemblance to the problem to be solved.* Analysis of compilation frequency and the ratio of all compiles to clean compiles may be useful here. Identifying code of little resemblance to the problem to be solved might involve static analysis, but looking at test failures on simple general cases might be just as effective.

- **Tracing stage (i.e. preoperational):** *When writing code, these novices tend to patch and repatch their code. These novices struggle to truly design a solution*; whereas ...
- **Post-tracing Stage (i.e. concrete operational):** *This stage is the first stage where students begin to show a purposeful approach to writing code*. Thus a method for the automatic detection of a "purposeful approach" is the key issue for distinguishing between tracing and post-tracing novices.

3. The Research Question

The current formulation of my research question is as follows:

> *Do novices who are reasoning at each of these three different neo-Piagetian stages go about writing programs differently, in ways automatically detectable by the computer they are using?*

The significance of this research is two-fold:

1) It will establish that neo-Piagetian stages do not just occur in "artificial" pencil and paper exercises, but also occur in the "natural" environment – that is, as novices write code on the computer.
2) If the neo-Piagetian stage of a student can be automatically identified, then a computer-based tutoring system could use that information. (However, the development of such a tutoring system is beyond the scope of my project.)

4. A Candidate for the Primary Data Collection Tool — Web-CAT

My most recent work has focussed on identifying a suitable, existing data collection tool for conducting a pilot. The current preferred data collection tool is Web-CAT, which is an established automated grading system that can be used to assess student's programs. Other researchers have already used Web-CAT to study the behaviour of novice programmers (Edwards, et al., 2009; Edwards, 2013). Web-CAT can store student software testing activities, in addition to simply judging student work by comparing program output to target output. Web-CAT is implemented as a web application with a well-defined API plug-in-style architecture so that it also can serve as a platform for providing additional student support and/or data collection. It supports static analysis tools to assess documentation and coding style and manual grading with direct on-line markup of assignments.

The students at my institution are first taught the Java programming language. Web-CAT supports the following features for processing Java programs:

- Support for static analysis of student code using Checkstyle to identify documentation, formatting, naming, and stylistic errors.
- Support for static analysis of student code using PMD to identify additional stylistic and coding errors.
- Support for optional instructor-provided reference tests that will be automatically executed against the student code.
- The ability to write new PMD checks using xpath expressions (or provide instructor's designed Java implementations of PMD or Checkstyle checks).

5. Granularity of Data

At this early stage of the PhD project, an issue under current consideration is the type of data I should collect for each student as they work on a specific programming problem. Some forms of data under consideration for collection are, from least demanding and sparse to more demanding and dense:

- **Every clean compile:** The amount of data collected would easily be manageable. However, this is the type of data that has been collected in many earlier systems and thus the novelty is low. Also, there is a risk that this type of data may not be enough to identify the different neo-Piagetian stages.

- **Every compile:** At this time, we suspect that this project will focus on distinguishing between students at the tracing and post-tracing stages, and we suspect that information about syntax errors is mostly an issue for pre-tracing students.
- **Every keystroke and mouse click:** While this is very rich data, it will pose data management issues. Also, it is not clear to us how we would use such data, apart from its simple use to estimate whether the student is actually working on the given programming task. One possibility is to look for a "rhythm" to user key presses and mouse clicks. We tentatively suspect that post-tracing (concrete operational) novices will display a steady "rhythm", while novices at the two earlier neo-Piagetian stages will display more sporadic behaviours.
- **Periodic Facial Images:** This is of interest for two reasons, assuming suitable software for analyzing the images already exists. First, such data would help establish that the novice under study is actually looking at the computer screen, and is not distracted. (Otherwise, time on task is hard to estimate automatically.) Second, the emotional state of the novice may be salient to detecting how well a student is coping with a given programming exercise (Good et al., 2011; Rodrigo and Baker, 2009). We suspect that post-tracing (concrete operational) novices will display more deliberate and systematic approaches to code writing and debugging, and will show less negative emotion than students at the two earlier neo-Piagetian stages. There already exists expertise at our university in analyzing facial images to detect emotions (Tan, Leong and Shen, 2014).
- **Eye-tracking data:** Eye-tracking data from novice programmers has already been extensively studied (e.g. Bednarik and Tukainine, 2004; Bednarik et al., 2005 & 2006; Nevalainen and Sajaniemi, 2004), but not from within a neo-Piagetian framework. We suspect that a pre-tracing (sensorimotor) novice student may either be focussed on the wrong part of the program, or may be glancing rapidly and perhaps randomly around the whole program. As our eventual aim (beyond this PhD) is to develop a tutoring system that is deployable on off-the-shelf computers, we expect we will only collect eye-tracking data if we can do so with easily accessible hardware and software.

Capturing data at every compile or every clean compile is easily compatible with the Web-CAT system, but the other options would require more sophisticated and more custom software.

Capturing facial images may present difficulties with getting clearance from our university's ethics committee.

6. Pilot Study (July – November, 2014)

In the coming southern hemisphere semester (i.e. July – November, 2014), we will conduct a pilot study in an introductory programming subject, which has over 300 enrolled students. A major component of the subject's assessment will be in the form of regular paper- and lab-based tests, held every second week. Furthermore, some of these tests are based upon the past work of Teague and Lister, and thus already have a neo-Piagetian component. The pilot will focus upon collecting and analyzing data from these tests, using Web-CAT.

The granularity of the pilot data collected from the introductory programming subject will be at the compile level. During this pilot phase, we will also evaluate the technical feasibility of collecting keystroke, facial and eye-tracking data, but we do not plan to collect such data with the aim of performing serious analysis during this pilot.

After the pilot, we expect that the "serious" data collection for the project will occur in the second year of PhD candidature, in the two southern hemisphere semesters of 2015 (i.e. Feb – June and July – November).

7. References

Ahadi, A. and Lister, R. (2013) *Geek genes, prior knowledge, stumbling points and learning edge momentum: parts of the one elephant*? In Proceedings of the ninth annual international ACM conference on International computing education research (ICER '13). ACM, New York, NY, USA, 123-128. http://doi.acm.org/10.1145/2493394.2493416

Ahadi, A., Lister, R. and Teague, D. (2014) *Falling Behind Early and Staying Behind When Learning to Program*. Paper presented at the 25th Anniversary Psychology of Programming Annual Conference (PPIG), Brighton, England , 25th-27th June 2014.

Bednarik, R. and Tukainine, M. (2004) *Visual attention and representation switching in Java program debugging: a study using eye movement tracking.* 16th PPIG Conference. pp. 159-169

Bednarik, R., Myller, N., Sutinen, E. & Tukiainen, M. (2005) *Effects of experience on gaze behaviour during program animation*. 17th PPIG Conference. pp 49-61.

Bednarik, R., Myller, N., Erkki Sutinen, E., and Tuki, M. (2006) *Program Visualization: Comparing Eye-Tracking Patterns with Comprehension Summaries and Performance*. 18th PPIG Conference. pp. 68 – 82

Brown,N., Kolling, M., McCall, D. And Utting, I. (2014) *Blackbox: A Large Scale Repository of Novice Programmer' Activity*. In Proceedings of the 45th ACM technical symposium on Computer science education (SIGCSE '14). pp. 223-228. http://doi.acm.org/10.1145/2538862.2538924

Corney, M., Teague, D., **Ahadi, A.,** & Lister, R. (2012). *Some Empirical Results for Neo-Piagetian Reasoning in Novice Programmers and the Relationship to Code Explanation Questions*. Paper presented at the 14th Australasian Computing Education Conference (ACE 2012). pp. 77-86. http://www.crpit.com/confpapers/CRPITV123Corney.pdf

Crosby, M. Scholtz, J. Wiedenbeck, S. (2002) *The Roles Beacons Play in Comprehension for Novice and Expert Programmers*. 14th PPIG Conference. pp. 58-73

Du Boulay, B. (1989). *Some Difficulties of Learning to Program*. In E. Soloway & J. C. Sphorer (Eds.), Studying the Novice Programmer (pp. 283-300). Hillsdale, NJ: Lawrence Erlbaum.

Edwards, S., Snyder, J., Pérez-Quiñones, M., Allevato, A., Kim, D. and Tretola, B. (2009) *Comparing effective and ineffective behaviors of student programmers*.In Proceedings of the fifth international workshop on Computing education research workshop (ICER '09). pp. 3-14. http://doi.acm.org/10.1145/1584322.1584325

Edwards, S. (2013) Continuous Data-driven Learning Assessment. In Future Directions in Computing Education Summit White Papers (SC1186). Dept. of Special Collections and University Archives, Stanford University Libraries, Stanford, Calif. http://www.stanford.edu/~coopers/2013Summit/EdwardsStephenVaTech.pdf

Ginat, D. (2007). Hasty design, futile patching and the elaboration of rigor. *SIGCSE Bull.* 39, 3 (June), 161-165. http://doi.acm.org/10.1145/1269900.1268832

Good, J., Rimmer, J. Harris, E. and Balaam, M. (2011) *Self-Reporting Emotional Experiences in Computing Lab Sessions: An Emotional Regulation Perspective*. 23rd PPIG Conference.

Jadud, M. (2006) *An Exploration of Novice Compilation Behaviour in BlueJ.* PhD thesis, University of Kent. http://kar.kent.ac.uk/14615/

Khan, I. A., Brinkman, W-P. and Hierons, R. (2008) *Towards a Computer Interaction-Based Mood Measurement Instrument.* 20th PPIG Conference.

Lister, R. (2011). *Concrete and Other Neo-Piagetian Forms of Reasoning in the Novice Programmer*. Paper presented at the 13th Australasian Computer Education Conference (ACE 2011). pp. 9-18. http://crpit.com/confpapers/CRPITV114Lister.pdf

Nevalainen, S. and Sajaniemi, J. (2004) *Comparison of Three Eye Tracking Devices in Psychology of Programming Research* .16th PPIG Conference. pp. 151-158

Norris, C., Barry, F., Fenwick Jr., J. B., Reid, K., and Rountree, J. (2008). *ClockIt: collecting quantitative data on how beginning software developers really work*. In *Proceedings of the 13th* *annual conference on Innovation and technology in computer science education* (ITiCSE '08). pp. 37-41. http://doi.acm.org/10.1145/1384271.1384284

Rodrigo, M. and 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). pp. 75-80. http://doi.acm.org/10.1145/1584322.1584332

Romero, P., du Boulay, B., Cox, R., Lutz, R. and Bryant,S. (2004) *Dynamic rich-data capture and analysis of debugging processes*. 16th PPIG Conference. pp. 140-150

Romero, P, du Boulay, B., Cox, R., Lutz, R & Bryant, S. (2005) Graphical visualisations and debugging: a detailed process analysis. 17th PPIG Conference. pp. 62 - 76

Tan, C., Leong, T. and Shen, S. (2014) *Combining think-aloud and physiological data to understand video game experiences*. In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '14). pp. 381-390. http://doi.acm.org/10.1145/2556288.2557326

Teague, D., Corney, M., **Ahadi, A.**, & Lister, R. (2012). *Swapping as the "Hello World" of Relational Reasoning: Replications, Reflections and Extensions*. Paper presented at the Australasian Computing Education Conference (ACE 2012). pp. 87-94. http://www.crpit.com/confpapers/CRPITV123Teague.pdf

Teague, D., Corney, M., Fidge, C., Roggenkamp, M., **Ahadi, A.**, & Lister, R. (2012). *Using Neo-Piagetian Theory, Formative In-Class Tests and Think Alouds to Better Understand Student Thinking: A Preliminary Report on Computer Programming*. Paper presented at the Australasian Association for Engineering Education Conference (AAEE 2012). http://www.aaee.com.au/conferences/2012/documents/abstracts/aaee2012-submission-22.pdf

Teague, D., Corney, M., **Ahadi, A.**, & Lister, R. (2013). *A Qualitative Think Aloud Study of the Early Neo-Piagetian Stages of Reasoning in Novice Programmers*. Paper presented at the 15th Australasian Computing Education Conference (ACE 2013).

Teague, D., & Lister, R. (2014a). *Longitudinal Think Aloud Study of a Novice Programmer*. Paper presented at the Australasian Computing Education Conference (ACE 2014). pp. 41-50. http://crpit.com/Vol148.html

Teague, D., & Lister, R. (2014b). *Manifestations of Preoperational Reasoning on Similar Programming Tasks*. Paper presented at the Australasian Computing Education Conference (ACE 2014). pp. 65-74. http://crpit.com/Vol148.html

Teague, D. and Lister, R. (2014c) *Programming: Reading, Writing and Reversing*. 19th Annual Conference on Innovation and Technology in Computer Science Education (ITiCSE'14), June 23-25, 2014, Uppsala, Sweden.

Teague, D. and Lister, R. (2014d) *Blinded by their Plight: Tracing and the Preoperational Programmer.* Paper presented at the 25th Anniversary Psychology of Programming Annual Conference (PPIG), Brighton, England, 25th-27th June 2014. (i.e. this conference)

Teague, D. (2014e) Neo-Piagetian Theory and the Novice Programmer. Doctoral Consortium paper presented at the 25th Anniversary Psychology of Programming Annual Conference (PPIG), Brighton, England, 25th-27th June 2014. . (i.e. this conference)

Winters, T. and Payne, T. (2005). What do students know?: an outcomes-based assessment system. In Proceedings of the First international Workshop on Computing Education Research (Seattle, WA, USA, October 1–2). ICER '05. ACM, New York, NY, 165–172.