Understanding the Novice Programmer

Trilby Hren
Supervising Faculty: Dr. Tom Allen, Computer Science & Dr. Temi Bidjerano, Education
Department of Computer Science, Furman University, Greenville, SC

FURMAN

Introduction

The computer science education literature suggests that the success rate in introductory programming courses is worse than hoped. While the problem has been recognized a long time ago, there is very little agreement among computer science educators about what should be done about it.

In spite of some useful research on novice programmers, customary programming pedagogy remains largely uninformed. Most of the research on novice programmers concludes that novice programmers (introductory students) think about programming very differently from expert programmers (their teachers). Consequently, pedagogy based on the concepts and mental models used by expert programmers is not likely to be effective for instructing novice programmers.

To develop and test potential interventions for teaching introductory programming, it is important to have a more detailed grasp of the typical conceptual or mental models that novice programmers apply to these learning tasks. It is also useful to have a more accurate assessment of how these conceptual/mental models develop as the student progresses through the programming sequence.

Objectives

Long-Term Objectives

- Describe how novice, intermediate, and advanced computer science students approach programing tasks in computer science with the use of a combination of quantitative and qualitative methodologies
- Describe progression in skill acquisition with the goal to develop better working models of relevant cognitive skills and learner characteristics
- Develop a blueprint for more effective instructional interventions

Objectives for Pilot Study

- Develop a preliminary typology of problem-solving skillsets
- Pilot quantitative and qualitative data collection procedures to identify potential challenges in the main study

Methods

Participants

- Ninety-one Furman students enrolled in a computer science course in Fall 2015
- Sampling from multiple sections of introductory (CS 105) , intermediate (CS 121 & Cs 122), and advanced (CS 341) computer science courses

Indirect Measure

- Three parallel forms of a paper and pencil test consisting of 10 short-answer and fill-in-the blank items
- Identical concepts assessed; code segments reflect different programming languages

Table 1. Sample Items

Q1. In the blanks below, supply the values for the variables *after* the following code has been executed.
`a = 15`
`b = 57`
`c = 22`
`a = b`
`c = a`
`b = c`
 	
The value of **a** is ____
The value of **b** is ____
The value of **c** is ____
	Q4. Consider the following Python function
	
	`# x is a list and val is a value`
	`def function_b(x, val):`
	`  answer = false`
	`  i = 0`
	`  while (i < len(x)):`
	`  if (x[i] == val):`
	`  answer = true`
	`  break`
	`  else:`
	`  i = i + 1`
	`  return answer`
	Explain (in plain English) the purpose of this function.

Skill assessed: Ability to trace code	*Skill assessed: Ability to explain code*

Direct Measure: Think Aloud

- The method involved asking each participant to think aloud while solving a computer science problem, documenting the process, and analyzing the resulting verbal account
- Individual sessions with 3 problems per participant
- Livescribe™ Smartpen used for data recording purposes; it allowed spoken words to be audio-recorded, automatically transcribed and saved as a digital text

Results

Typology of Skillsets

- Latent Class Analysis (LCA) - a person-centered method used to explains the variability in responses as a function of membership to an unobserved, hypothetical group called a *latent class*.
- Participants are assigned to unique homogeneous latent classes based on similar arrays of correct or incorrect responses.
- Members of each latent class have similar skillsets or characteristics; clusters maximize/augment differences

Figure 1. Results from LCA – Three Class Solution

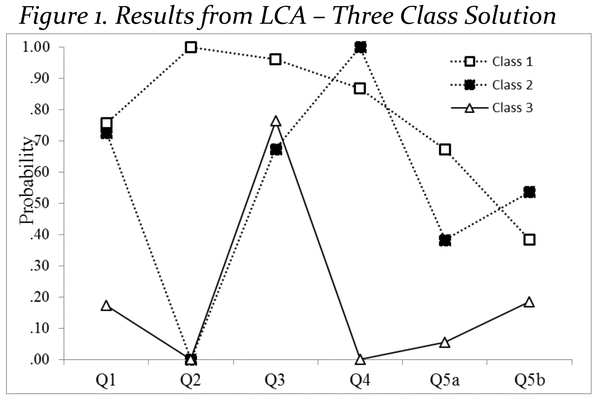


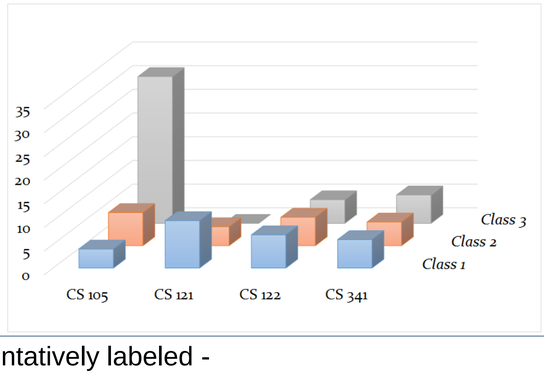
Table 2. Results from LCA – Three Class Solution

Test Items	Overall	Best Solution		
		Class 1	Class 2	Class 3
Code tracing [Q1]	.48	.76	.73	.17
Writing code [Q2]	.34	1.00	.00	.00
Explaining code [Q3]	.80	.96	.67	.76
Explaining code [Q4]	.51	.87	1.00	.00
Debugging code [Q5a]	.34	.67	.38	.06
Debugging code [Q5b]	.34	.38	.54	.18
N	91	27	22	42
Percent	100	29.67	24.18	46.15

Probability of a correct response to an item for members belonging to a given latent class.

- The relationship between latent class membership and type of course was examined in a series of follow-up analyses. Results indicate that students in the introductory sections of CS 105 were significantly more likely ($p < .05$) to belong to the low-skill latent class

Figure 2. Course by Latent Class Distribution



Conclusion

We identified three subgroups of computer science students based on skillsets. These were tentatively labeled -

- *Writers* (LC 1) – students with relatively well-developed skills in all areas assessed
- *Explainers* (LC 2) – students deficient in writing code, but shortcoming in writing code not affecting other areas of programming competence
- *Novices* (LC 3) – a low-skill group dominated by students enrolled in the introductory computer science course

All students were likely to experience difficulty on the debugging tasks (which presumably entail higher order programming skills) but this tendency tends to be less pronounced among *Writers* and *Explainers*.