Abstract
Computer science educators agree that, for many beginners, learning to write computer programs is very difficult. And, in spite of our best efforts at remedying the situation, many novice programmers still struggle. We report some preliminary findings from our study that seeks to understand what sort of preconceptions novices use when learning to program. Our early results show unsurprisingly that student programmers can be separated distinctly into the haves and have-nots. This separation seems based on mastery of programming skills that are likely developed in an incremental fashion that suggests a hierarchy. But, in spite of these findings, we have also noted that predicting whether a student can correct or debug a program does not seem to depend on their success in these other areas.

1. Introduction
It has been fifteen years since the first McCracken group concluded that introductory programming students completing their first course do not “know how to program at the expected skill level.” (McCracken, 2001) No doubt many of us had experienced this anecdotally, but the report suggested that the situation was both widespread and measurable.


Experimenting with languages, courses, and teaching methods is useful but most of these efforts sidestepped a more fundamental issue stated over thirty years ago by Spohrer and Soloway,

A reasonable pedagogical philosophy is The more we know about what students know, the better we can teach them (their emphasis). Yet in the past, what educators knew about novice programmer was largely “folklore”: anecdotal evidence from their own and their colleagues’ experiences. (Spohrer, 1986)

In short, do novice programmers (introductory students) think about programming in the same manner as expert programmers (their teachers)? The research on novice programming over the past fifteen years is united on one point: novice programmers are different from expert programmers in how they think about and approach problem-solving using programming. (e.g., Hofer, 2010, Jimoyiannis, 2011, Lahtinen, 2007, Lui, 2006, Ramalingam, 1997, Weiser, 1983, Ye, 1996). Defining the nature of these differences is another matter.

Some of the more interesting and extended studies have been conducted by researchers who were originally associated with the BRACElet project. (e.g., Clear, 2008, Clear, 2009, Clear, 2011, Tan, 2010, Whalley, 2009) A significant outcome of their research is that writing programs is a higher-order skill that is built upon the acquisition of more basic cognitive skills (Lister, 2009, Lopez, 2008, Venables, 2009). Introductory students cannot be expected to write programs effectively until they master these other skills. Several hierarchical learning taxonomies have been applied to help explain this (e.g., Corney, 2012, Lister, 2000, Lister, 2006, Teague, 2012, Teague, 2015, Whalley, 2006).
2. Research Agenda
Our ultimate goal is to develop and test more effective interventions for teaching introductory programming. But, first, it is imperative to have a better grasp of the typical conceptual or mental models that novice programmers apply to these learning tasks. It would also be useful to have a more accurate assessment of how students’ conceptual/mental models develop as they progress through the programming sequence.

Furman University is a small, selective liberal arts college in the U.S. with an enrollment of 2,800 students. We offer three degrees in computing and currently have 60+ majors. Consequently, our subjects represent both a smaller and perhaps different sample compared to those of previous studies.

The major objectives of our study are these:

• to replicate (repeat) and extend previous research on novice vs. expert programmers (Lister, 2008).
• to contribute to a more informed model of the novice programmer in order to investigate effective pedagogical interventions.

We hope to determine whether the performance of novices conforms to what traditional research tells us (e.g., Kuolari, 2015, Robins, 2003). In short, novices focus on syntactic details of code in terms of line-by-line execution. They lack viable mental models for problem-solving and programming. In contrast, more expert programmers understand programs based on abstracting parts of the code and have a variety of mental models for problem-solving.

Our focus will be on finding the dominant mental models that novices develop for programming solutions using variables and assignment statements, control structures such as selection and repetition, as well as code writing and debugging strategies.

3. Methods
Many of the previous studies on novice programmers collected evidence from testing novice programmers using standard instruments such as multiple-choice tests. While it is possible to assess the correctness of answers, it is not clear how or why the students selected their responses. Thus, the researcher can only hypothesize what sort of mental models students employed. If the research is accurate, then expert programmers think very differently than novice programmers when solving problems. As a result, instructors (as expert programmers) would find it difficult to imagine how their students (as novice programmers) think. Recently, though, think-aloud methodology has been used for more direct evidence about how students go about solving programming problems (Teague, 2012, Teague, 2014, Van Someren, 1994, Whalley, 2014).

Over the past year, we have employed both traditional testing for larger groups as well as recording think-aloud sessions with a smaller number of students.

4. Preliminary Results
To achieve a better understanding of the kinds of skills that qualify distinctions among students, we attempted to develop a typology of students based on item response patterns from a short written test. Most of the questions were derived from or based on those employed in previously reported research (Denny, 2012, Ramalingam, 1997, Teague, 2012, Teague, 2015). The intent was to compare our results with those from other studies. We also added questions in order to assess basic (logical) debugging skills.

We sought to identify subsets of students with a similar pattern of correct and incorrect answers to the computing tasks posed by the test questions. To this end, we used Latent Class Analysis (LCA). LCA is a person-centered method used to explain variability of responses as a function of membership to unobserved but assumed existing groups. Sample members are assigned to unique homogeneous latent classes based on similar arrays of correct or incorrect responses. Members of each latent class are more similar than dissimilar and clusters are formed in such a way that differences between clusters are augmented or maximized (Muthen, 2000). Using this statistical approach, we identified two clusters of students with unique response patterns. The two-latent class model was superior to a
one-class solution (Lo-Mendell-Rubin Adjusted LL = 181.15, \( p < .001 \)) as well as the best fitting model in comparison to all other models considered. The entropy statistic (a measure of classification accuracy) of the two-class model was .86. The number of cases were evenly distributed between classes with about half of the students identified as belonging to either LC1 (n=71, 49%) or LC2 (n = 74, 51%). Results are presented in Table 1.

The first column in Table 1 shows the proportion of students in the sample who were able to provide a correct response to each of the problems given. We see that nearly all students answered one of the tracing questions (i.e., T2) correctly whereas less than 50% of the sample succeeded on the problems involving tracing code (T1, 47%), explaining code (E2, 47%). The remaining three columns in Table 1 display the conditional probability of a correct response for a member belonging to a given class. As seen, the likelihood for a student belonging to LC1 of being correct on items assessing tracing, writing, and explaining code is quite high. By contrast, a student assigned to LC2 has lower chances of having a successful attempt on the same types of items. Overall, compared to LC2 students, LC1 students were much more likely to produce correct solutions to a variety of item types; at the same time, the success rate across items is notably lower for LC2 students. LC2 students appear to be particularly deficient with respect to writing code (W1 and W2) and explaining code (E2). The findings suggest also that the most reliable differences between the identified categories lie in the area of debugging. A LC1 student is four to five times as likely to answer the first (DB1: Odds Ratio = 4.79, \( p = .028 \)) and second debugging item (DB2: Odds Ratio = 4.37, \( p = .02 \)) correctly as is a LC2 student. Differences between latent classes are depicted in Figure 1.

<table>
<thead>
<tr>
<th>Test Items</th>
<th>Overall Proportion</th>
<th>Two-Class Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Class 1</td>
</tr>
<tr>
<td>Trace [T1]</td>
<td>.469</td>
<td>.823</td>
</tr>
<tr>
<td>Write [W1]</td>
<td>.371</td>
<td>.619</td>
</tr>
<tr>
<td>Explain [E1]</td>
<td>.793</td>
<td>.861</td>
</tr>
<tr>
<td>Explain [E2]</td>
<td>.465</td>
<td>.838</td>
</tr>
<tr>
<td>Debug [D1]</td>
<td>.382</td>
<td>.552</td>
</tr>
<tr>
<td>Debug [D2]</td>
<td>.374</td>
<td>.534</td>
</tr>
<tr>
<td>Write [W2]</td>
<td>.391</td>
<td>.806</td>
</tr>
<tr>
<td>Trace [T2]</td>
<td>.928</td>
<td>.963</td>
</tr>
<tr>
<td>Trace[T3]</td>
<td>.816</td>
<td>.877</td>
</tr>
<tr>
<td>Trace [T4]</td>
<td>.618</td>
<td>.669</td>
</tr>
<tr>
<td>N</td>
<td>145</td>
<td>71</td>
</tr>
<tr>
<td>Percent</td>
<td>100%</td>
<td>49%</td>
</tr>
</tbody>
</table>

Table 1: Results from Latent Class Analysis
Results from a chi-square test of independence estimating the relationship between type of course and latent class membership indicated that the students in the pre-programming course (CS0) were more likely to belong to LC2, $\chi^2_{145} = 58.25, p < .001$. Additional analyses – unrelated to focal questions of interest – revealed that (a) GPA was positively correlated with LC1 probability, $r_{52} = .31, p = .026$, suggesting that students who were stronger academically were more likely to be part of LC1; (b) gender did not have a measurable effect on the likelihood of belonging to either class, $t_{50} = .31, p > .05$; and (c) LC1 students tended to find the items easier in greater proportions as shown by a moderate positive correlation between number of items endorsed as being easy and latent class probabilities, $r_{30} = .57, p = .001$.

![Figure 2: Distribution of students in latent classes by course level.](image)

Our latent class analysis demonstrates that, regardless of classification, students tend to be only moderately successful on computing problems requiring debugging. While in regards to the category of LC2 students this finding does not seem to be surprising, the somewhat lower probabilities for success among the abler LC1 subgroup suggest that debugging is relatively independent of other skillsets. This observation is consistent with results from additional correlation analyses showing weak correlations between the two debugging items in the assessment and other skills. Performance on the tracing sub-questions (based on the Parson’s problem) was virtually unrelated to performance on the two debugging items (DB1 & DB2) as documented by small, non-significant correlations in the range from -.01 to .26. The correlations between the first tracing item (T1) and DB1 ($r_{143} = .20, p < .05$) and T1 and DB2, ($r_{143} = .267, p < .001$) were statistically significant, yet small. The strongest associations were between DB1 and E2 ($r_{143} = .37, p < .001$) and between DB1 and W2 ($r_{143} = .32, p < .001$). The corresponding coefficients between DB2, E2, and W2 were even smaller ($r_{143} = .27$ and $r_{143} = .31$, respectively). Overall, we failed to uncover evidence of co-variation between ability to correct error in code and other computing skills. Although somewhat related to other essential skills, ability to debug appears to be sufficiently distinct.

### 5. Future Work

During the summer (2016), we plan to code and analyze the 30+ hours of audio/video collected from the think-aloud sessions of our twelve subjects. Each subject completed four separate sessions. These included problems involving assignment statement, selection and iteration control structures. Subjects were asked to trace written code, predict outcomes, supply missing statements from code segments, read and explain written code segments, fix (mostly) logical errors in supplied code segments, and write a short program. Most of the work was done using paper and pencil, but the latter two activities were completed using a computer for feedback. The preliminary results suggest the subjects had varying levels of skills and abilities in these areas. In addition, the subjects’ approach to problem-solving often differs significantly when the work is done using a computer. We expect that these sessions will provide a rich source for qualitative analysis.

At any rate, these results will provide a basis for the next stage of our project. However, based on our preliminary results, exploring the interplay of writing and debugging programs appears to be a useful avenue of further investigation.
6. References


