Work In Progress Report: Tracking the Novice Programmer

J. Thomas Allen
Department of Computer Science
Furman University
tom.allen@furman.edu

Temi Bidjerano
Department of Education
Furman University
temi.bidjerano@furman.edu

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.

Many computer science educators responded by experimenting with new teaching methods (e.g., Ambrosio, 2011, Barg. 2000, Beck, 2013, Campbell, 2014, Chong, 2005, Gaspar, 2012, Grissom, 2013, Hu, 2013, Hu, 2014, Radinski, 2006, Salleh, 2011, Wood, 2013), different programming languages (Guo, 2014, Koulori, 2015), paradigms (Vujosevic, 2008), and new ways of fashioning the introductory course (Guzdial, 2003, Hundahausen, 2008, O'Kelly, 2006, Sung, 2008, Xu, 2008). But, in spite of these efforts, students' success rates have not improved much (Watson, 2014). This prompted some to question these approaches (Gardner-McCune, 2013, Guzdial, 2015, Lister, 2008a, Randolph, 2008, Siegfried, 2008, Valentine, 2004).

Experimenting with languages, courses, and teaching methods is useful but most of these efforts side-stepped 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, 2008b).
- 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., Kuolori, 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.

Test Items	Overall	Two-Class Solution	
	Proportion	Class 1	Class 2
Trace [T1]	.469	.823	.112
Write [W1]	.371	.619	.030
Explain [E1]	.793	.861	.721
Explain [E2]	.465	.838	.068
Debug [D1]	.382	.552	.204
Debug [D2]	.374	.534	.208
Write [W2]	.391	.806	.038
Trace [T2]	.928	.963	.897
Trace[T3]	.816	.877	.762
Trace [T4]	.618	.669	.573
N	145	71	74
Percent	100%	49%	51%

Table 1: Results from Latent Class Analysis

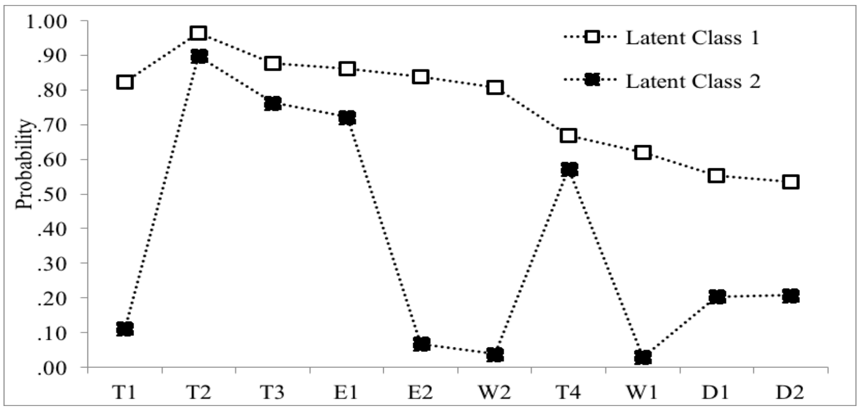

Figure 1: Probability of correct solution by latent class

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, $r_{30} = .57$, $p = .001$.

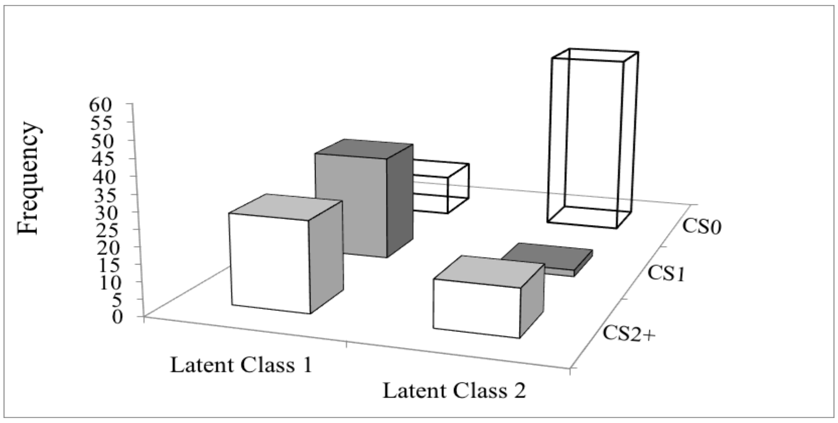

Figure 2: Distribution of students in latent classes by course level.

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

Ambrosio, Ana Paula, et al. (2011) "Identifying Cognitive Abilities to Improve CS1 Outcome," *Forty-first ASEE/IEEE Frontiers in Education Conference*, pp. F3G1–7.

Barg, Michael, Alan Fekete, et al. (2000) "Problem-Based Learning for Foundation Computer Science Courses," *Computer Science Education*, Vol. 10, No. 2, pp. 109-128.

Beck, Leland, and Alexander Chizhik. (2013) "Collaborative Learning Instructional Methods for CS1: Design, Implementation, and Evaluation," *ACM Transactions of Computing Education*, Vol. 13, No. 3, pp. 10:1–21.

Campbell, Jennifer, Diane Horton, et al. (2014) "Evaluating an Inverted CS1," *SIGSCE'14*, pp. 307–312.

Chong, Jan, Robert Plummer, et al. (2005) "Pair Programming: When and Why it Works," *Seventeenth Workshop of the PPIG*, pp. 43–48.

Clear, Tony, Jenny Edwards, Raymond Lister, et al. (2008) "The Teaching of Novice Computer Programmers: Bringing the Scholarly-Research Approach to Australia," *CPRIT*, Vol. 78, pp. 63–68.

Clear, Tony, Raymond Lister, Simon, et al. (2009) "Naturally Occurring Data as Research Instrument: Analyzing Examination Responses to Study the Novice Programmer," *ACM SIGSCE Bulletin*, Vol. 41, No. 4, pp. 156–173.

Clear, Tony, Jacqueline Whalley, Phil Robbins, et. al. (2011) "Report of the Final BRACElet Workshop," *Journal of Applied Computing and Information Technology*, Vol., 15, Issue 1.

Corney, Malcolm, Donna Teague, Alireza Ahadi, and Raymond Lister. (2012) "Some Empirical Results for Neo-Piagetian Reasoning in Novice Programmers and the Relationship to Code Explanation Questions," *CPRIT'12*, pp. 77–86.

Denny, Paul, Andrew Luxton-Reilly, and Beth Simon. (2008) "Evaluating a New Exam Question: Parson's Problems," *ICER'08*, pp. 113–124.

Gardner-McCune, Christina. (2013) "A Case for Learning Research in Computer Science Education," *Future Directions in Computing Education Summit*, pp. 1–4.

Gaspar, Alessio, and Sarah Langevin. (2012) "An Experience Report on Improving Constructive Alignment in An Introduction to Programming," *Journal of Computing Sciences in Colleges*, Vol. 28, No. 2, pp. 132–140.

Grissom, Scott, Beth Simon, et al. (2013) "Alternatives to Lecture: Revealing the Power of Peer Instruction and Cooperative Learning," *SIGSCE'13*, pp. 283–284.

Guo, Philip. (2014) "Python is Now the Most Popular Introductory Teaching Language at Top U.S. Universities," *BLOG@ACM*, http://cacm.acm.org/blogs/blog-cacm/176450-python-is-now-the-most-popular-introductory-teaching-language-at-top-us-universities.

Guzdial, Mark. (2003) "A Media Computation Course for Non-Majors," *ITiCSE'03*, pp. 104–108.

Guzdial, Mark. (2015) "What's the Best Way to Teach Computer Science to Beginners?" *CACM*, Volume 58, No. 2, pp. 12-13.

Hofer, Andreas. (2010) "Exploratory Comparison of Expert and Novice Pair Programmers," *Computing and Informatics*, Vol. 29, 73–91.

Hu, Helen, and Tricia D. Shepherd. (2013) "Using POGIL to Help Students Learn to Program," *ACM Transactions on Computing Education*, Vol. 13, No. 3, pp. 13:1–23.

Hu, Helen, and Tricia D. Shepherd. (2014) "Teaching CS1 with POGIL Activities and Roles," *SIGSCE'14*, pp. 127–132.

Hundahausen, Christopher D., and Jonathan Lee Brown. (2008) "Designing, Visualizing, and Discussing Algorithms within a CS1 Studio Experience: An Empirical Study," *Computers & Education*, Vol. 50, No. 1, pp. 301–326.

Jimoyiannis, Athanassios. (2011) "Using SOLO Taxonomy to Explore Students' Mental Models of the Programming Variable and the Assignment Statement," *Themes in Science & Technology Education*, Vol. 4, No. 2, pp. 53–74.

Koulori, Theodora, Stanislao Lauria, and Robert D. Macredie. (2015) "Teaching Introductory Programming: A Quantitative Evaluation of Different Approaches," *ACM Transactions on Computing Education*, Volume 14, No. 4, pp. 26:1–28.

Lahtinen, Essi. (2007) "A Categorization of Novice Programmers: A Cluster Analysis Study," *Fifteenth Workshop of the PPIG*, pp. 32–40.

Lister, Raymond. (2000) "On Blooming First Year Programming and its Blooming Assessment," *ACE'00*, pp.158–162.

Lister, Raymond, Beth Simon, Errol Thompson, et al. (2006) "Not Seeing the Forest for the Trees: Novice Programmers and the SOLO Taxonomy," *ACM SIGCSE Bulletin*, Vol. 38, No. 3, pp. 118–122.

Lister, Raymond. (2008a) "After the Gold Rush: Toward Sustainable Scholarship in Computing," *Proceedings of the Tenth Conferences on Australasian Computing Education*, Vol. 78, pp. 3–17.

Lister, Raymond. (2008b) "The Originality Glut," *Inroads—SIGSCE Bulletin*, Vol. 40, No. 2, 14–15

Lister, Raymond. (2009) "Further Evidence of a Relationship between Explaining, Tracing, and Writing Skills in Introductory Programming," *ACM SIGSCE Bulletin*, Vol. 41, No. 3, pp. 161–165.

Lopez, Mike, Jacqueline Whalley, Phil Robins, and Raymond Lister. (2008) "Relationships Between Reading, Tracing and Writing Skills in Introductory Programming," *ICER'08*, pp. 101–111.

Lui, Kim Man, Keith C.C. Chan, (2006) "Pair Programming Productivity: Novice-Novice vs. Expert-Expert," *International Journal of Human-Computer Studies*, Vol. 64, pp.915–925.

McCracken, Michael et al. (2001) "A Multinational, multi-institutional study of assessment of programming skills of first-year CS students," *ACM SIGSCE Bulletin*, Vol. 33, No. 4, pp. 125-140.

Muthen, B. and L.K. Muthen. (2000) "Integrating Person-Centered and Variable-Centered Analyses: Growth mixture modeling with latent trajectory Classes," *Alcohol Clinical Experimental Research*, pp. 882–891.

O'Kelly, Jackie, and J. Paul Gibson. (2006) "RoboCode & Problem-Based Learning: A Non-Prescriptive Approach to Teaching Programming," *ITiCSE'06*, pp. 217–221.

Radenski, Atanas. (2006) "'Python First': A Lab-Based Digital Introduction to Computer Science," *ITiSCE'06*, pp.197–201.

Ramalingam, Venilla, and Susan Wiedenbeck. (1997) "An Empirical Study of Novice Program Comprehension in the Imperative and Object-Oriented Styles," *ESP '97*, pp. 124–139

Randolph, Justus, George Julnes, et al. (2008) "A Methodological Review of Computer Science Education Research," *Journal of Information Technology*, Vol. 7, pp. 135–162.

Robins, Anthony, Janet Rountree, and Nathan Rountree. (2003) "Learning and Teaching Programming: A Review and Discussion," *Computer Science Education*, Vol. 13, No. 2, pp. 137–172.

Salleh, Norsaremah, Emilia Mendes, and John C. Grundy. (2011) “Empirical Studies of Pair Programming for CS/CE Teaching in Higher Education: A Systematic Literature Review,” *IEEE Transactions on Software Engineering*, Vol. 37, No. 4, pp. 509–520.

Siegfried, R. M., D. Chays, and K. G. Herbert. (2008) “Will There Ever Be Consensus on CS1?” *Proceedings of 2008 International Conference on Frontiers of Education: Computer Science and Computer Engineering—FECS’08*, pp. 18–23.

Spohrer, James, and Elliot Soloway. (1986) “Novice Mistakes: Are the Folk Wisdoms Correct?” *CACM*, Vol. 29, No. 7, pp. 624–632.

Sung, Kelvin, Michael Panitz, et al. (2008) “Assessing Game-Themed Programming Assignments for CS1/2 Courses,” *GDCSE’08*, pp. 51–55.

Tan, Grace, and Anne Venables. (2010) “Wearing the Assessement ‘BRACElet’,” *Journal of Information Technology Education: Innovations in Practice*, Vol. 9, pp. IIP 25–34.

Teague, Donna et al. (2012) “Using Neo-Piagetian theory, Formative In-class Tests and Think Alouds to Better Understand Student Thinking: A Preliminary Report on Computer Programming”, *Proceedings of AAEE 2012 Conference*, pp. 772–780.

Teague, Donna, and Raymond Lister. (2014) “Longitudinal Think Aloud Study of a Novice *Programmer. Conferences in Research and Practice in Information Technology* (*CRPIT*), Vol. 148, pp. 41–50.

Teague, Donna. (2015) *Neo-Piagetian Theory and the Novice Programmer*, Thesis by Publication at the Queensland University of Technology.

Van Someren, Maarten W., Yvonne F. Barnard, and Jacobijn A.C. Sandberg. (1994) *The Think Aloud Method: A Practical Guide to Modelling Cognitive Processes*. London: Academic Press.

Venables, Anne, Grace Tan, and Raymond Lister. (2009) “A Closer Look at Tracing, Explaining, and Code Writing Skills in the Novice Programmer,” *ICER’09*, pp. 117–128.

Vujosevic-Janicic, and Dusan Tosic. (2008) “The Role of Programming Paradigms in the First Programming Courses,” *The Teaching of Mathematics*, Vol. XI, No. 2, pp. 63–83.

Valentine, David W. (2004) “CS Educational Research: A Meta-Analysis of SIGSCE Technical Symposium Proceedings,” *SIGSCE’04*, pp.255–259.

Watson, Christopher, and Frederick W.B. Li. (2014) “Failure Rates in Introductory Programming Revisited,” *ITiSCE’14*, pp. 39–44.

Weiser, Mark, and Joan Shertz. (1983) “Programming Problem Representation in Novice and Expert Programmers,” *International Journal of Man-Machine Studies*, Vol. 19, No. 4, pp. 391–398.

Whalley, Jacqueline, Raymond Lister, Errol Thompson, et al. (2006) “An Australasian Study of Reading and Comprehension Skills in Novice Programmers Using the Bloom and SOLO Taxonomies,” *ACE’06*, Vol. 52, pp. 243–252.

Whalley, Jacqueline, and Raymond Lister. (2009) “The BRACElet 2009.1 (Wellington) Specification,” *CRPIT*, Vol. 95, pp. 9–18.

Whalley, Jacqueline, and Nadia Kasto. (2014) “A Qualitative Think-Aloud Study of Novice Programmers’ Code Writing Strategies,” *ITiCSE’14*, pp. 279–84.

Wood, Krissi, et al. (2013) “It’s Never Too Early: Pair Programming in CS1,” *Proceedings of the Fifteenth Conferences on Australasian Computing Education*, Vol. 83, pp. 13–20.

Xu, Dianna, Douglas Blank, and Deepak Kumar. (2008) “Games, Robots, and Robot Games: Complementary Contexts for Introductory Computing Education,” *GDCSE’08*, pp. 66–70.

Ye, Nong, and Gavriel Salvendy. (1996) “Expert-Novice Knowledge of Computer Programming at Different Levels of Abstraction,” *Ergonomics*, Vol. 39, No. 3, pp. 461–481.