Beyond Introductory Programming: Success Factors for Advanced Programming

Arthur Hoskey
arthur.hoskey@farmingdale.edu

Paula San Millan Maurino
paula.maurino@farmingdale.edu

Farmingdale State College
State University of New York
Farmingdale, NY 11735

Abstract

Numerous studies document high drop-out and failure rates for students in computer programming classes. Studies show that even when some students pass programming classes, they still do not know how to program. Many factors have been considered to explain this problem including gender, age, prior programming experience, major, math background, personal attributes, and the programming language itself. Research in this area has mainly been confined to introductory programming courses. This study explores the problem at a higher level. It tracks students longitudinally as they move from the first introductory programming class, to the second introductory class, and finally, to completion of an advanced programming course. The research question answered was: What are the factors contributing to the success or lack of success in advanced programming? The success factors examined were the introductory programming language taken, number of programming classes taken, track (concentration in the major), math and logic background, time lapse between the introductory and advanced programming class, instructor, gender, and general GPA. The factors that influenced student success were found to be the introductory programming language, time lapse between the introductory and advanced class, general grade point average, and track. Identification of these factors will help educators to make the best decisions on how to improve computer curriculum and programs and help students become better programmers.

Keywords: programming, programming languages, programming success, programming failure, success factors

1. INTRODUCTION

Farmingdale State College, a campus of the State University of New York, is a four year college specializing in applied science and technology. The college has had in place a Bachelor of Science Degree in Computer Programming and Information Systems for the past eight years. The degree is offered by the Computer Systems Department in the School of Business at the college and has five tracks (concentrations within the major): networking, database, systems, programming, and web development. All students are required to take two semesters of programming at an introductory level. They are currently offered a choice of C++ or Visual Basic. In addition, they are

all required to take an additional upper level programming course in Java. All students must achieve a "C" or better in both introductory programming classes to enter the advanced Java class.

Professors teaching the advanced course have found that some students entering the advanced class do not have the entry level programming skills needed to succeed in the upper-level class. Many possible explanations have been offered for this problem. It has been suggested by some faculty members that students wait too long to take the advanced course and as a result, have forgotten what they learned in the introductory classes. Others state that it is difficult for students to switch languages and recommend that all three courses use the same language. Still others state it is the introductory language that is at fault. They feel that Visual Basic is not an appropriate language for teaching programming and should be dropped from the curriculum or offered only as an elective. Some wonder if the fact that students do not do well in the required math courses or put off taking them could be related. Finally, others state that only students in the programming track do well in the course. Perhaps students in the other tracks should not have to take the advanced course.

This study was an exploration of this problem. We wanted to identify the factors involved in the apparent loss or lack of programming ability experienced by some students as well as the factors leading to success for others. Once these factors are identified, we will be able to make the best decisions on how to improve the program and help our students become better programmers. As such, our research question was: What are the factors contributing to the success or lack of success in advanced programming?

2. LITERATURE REVIEW

Failure/Drop Out Rates

As we searched the literature, we immediately realized we were not alone. Numerous studies document high drop out and failure rates for programming students (Guzdial & Soloway, 2002; McKinney & Denton, 2004). In a worldwide study, Bennedsen & Caspersen (2005) found that 33% of students fail CS1.

Compounding the problem, some students pass, but do not actually learn to program. In a multi-national, multi-institutional study of assessment of programming skills of first year CS students, students averaged only 22.89 out of a possible expected 110 points (McCracken, Kolikant, Almstrum, Laxer, Diaz, Thomas, Guzdial, Utting, Hagan, & Wilusz, 2001). In a later study that built on the McCracken work, it was found that many students lacked the knowledge and skills that are a precursor to problem solving. They cannot read or systematically analyze a short piece of code (Lister, Adams, Fitzgerald, Fone, Hamer, Lindholm, Mc Cartney, Mostrom, Sanders, Seppala, Simon & Thomas, 2004).

Introductory Programming

Most of the literature in this area was confined to studying the problems encountered by students in introductory classes. The students in our research study have already completed two semesters of computing. Yet, some of these students appear to have the "shallow and superficial skills" described in a 2005 study of novice programmers by Lewandowski, Gutschow, McCartney, Sanders, & Shinners-Kennedy. In an international study of 500 students and teachers, Lahtinen, Ala-Mutka, & Jervinen (2005) found that the biggest problem of novice programmers is not the understanding of basic concepts, but rather learning to apply them.

Math/Prior Programming Experience

Many studies seeking to predict achievement in introductory programming courses have examined math background, previous programming experience, and previous academic background. Previous experience with programming and a math background seem to be positively related to success in introductory programming (Byrne & Lyons, 2001; Bennedsen & Casperson, 2005; Wilson & Shrock, 2001; Rountree, Rountree, Robins & Hannah, 2004). Once again, our students have completed two semesters of programming already. They are required to take calculus, but this is not a prerequisite for any of the programming classes. Some students procrastinate and put it off. Others need math remedial classes and cannot take it until those courses have been completed.

Other Personal Attributes

Some studies have looked at factors such as sex and age. These demographics do not seem to affect success in programming al-

though the numbers of females entering programming is much lower (Bennedsen & Caspersen, 2005; Byrne & Lyons, 2001; Wilson & Shrock, 2001). Other studies have attempted to link programming success with a student's grades in previous coursework, self-efficacy, "comfort level" , or motivation to get an "A" in the course (Wilson & Shrock, 2001; Bennedsen & Caspersen, 2005; Rountree, Rountree, & Robins, 2001; Wiedenbeck, 2005).

Programming Language

Other studies looked at the programming language used in the classroom. Of these, some analyzed the languages for their teaching efficacy (Mannila, Peltomaki, & Salakoski, 2006; Mannila & de Raadt, 2006; Chen, Monge, & Simon, 2006; Dehinbo, 2006; Russell, Russell, Pollacia & Tastle, 2009; McIver & Conway, 1996) and others looked at the reasons colleges selected a particular language (Parker, Chao, Ottaway & Chang, 2006; Bhatnager, 2009).

There was no consensus on the best language to use. Lahtinen, Ala-Mutka, & Jervinen (2005) found that the teaching language did not seem to affect the learning situation. Chen, Monge, & Simon (2006) concurred. However, McIver & Conway (1996) found that a substantial part of the difficulty encountered in programming classes arises from the structure, syntax, and semantics of the particular programming language used. Further, Mannila, Peltomaki & Salakoski (2006) found that students did just as well learning a simple language and then moving on to a more complex one. They also found that the best languages to use in teaching programming were the languages designed with teaching in mind. They agreed with other researchers, however, that language is selected for many reasons beyond pedagogical benefit. In a study of employers and educators by Bhatnagar (2009), the teaching of more than one language was recommended.

Major

Lastly, some studies looked at the student's major. Prasad & Li (2004) tried to determine if there were differences between students majoring in computing and those majoring in information systems enrolled in the same computer programming course. They noted that information systems students had a little more difficulty with C++, but that the difference was slight. A student's major or intended major was found to be insignificant in a study done by Bennedsen & Caspersen (2005). Rountree, Rountree & Robbins (2001) found no difference in success rates for 472 students in an introductory programming class in Java for computer science majors, information science majors, or non-computer majors.

3. METHODOLOGY

Farmingdale State College's school records were used to create a database containing information about all two hundred students who took Java Programming from 2005 through the fall 2009 semester. After the statistical analysis for the years 2005-2009 was complete, we added the results for the spring 2010 semester. The spring 2010 Java class included 25 additional grades. The final database contained two hundred and twenty-five grades for Java. These final grades constituted our measure of success in the class. The statistical analysis was performed on the database of two hundred and twenty-five students unless indicated otherwise.

The database held information on each student in the following areas:

- The programming language taken in the introductory classes
- Whether or not a logic class was taken before the introductory programming class
- The number of programming classes taken
- Grades in the programming classes
- Overall GPA
- Time elapsed between the introductory programming classes and the Java class
- The particular professors teaching the programming classes
- Major or track (concentration within the Computer Systems Department)
- The type and sequence of math courses taken

Statistical analysis was performed on the data to determine relationships, if any, between the variables and student success in the advanced Java course. As mentioned previously, success in the Java course was measured by the student's final grade. In particular, we wanted to determine the following

- Did addition of a logic course to the curriculum increase success in programming?
- Did the particular faculty member teaching the introductory course affect student success in the advanced Java course?

- Was there a difference in male and female success rates in the Java class?
- Were students who took more than the minimum number of programming courses more successful in the advanced Java course?
- Did taking the required calculus course before Java increase success?
- Did the amount of time lapsed between taking the advanced Java course and completion of the introductory courses affect success in the Java class?
- Did students who took Visual Basic in the introductory courses do better or worse in the advanced Java class than students who took C++?
- Did students with a higher general GPA achieve greater success in the Java class?
- Did students in the programming track perform better in Java than students in the systems, web development and networking tracks?

4. RESULTS

Overview

A summary of our results appears in table 1 below.

Table 1

Summary of Study Results

Independent Variable	Difference In Java Grades?
Time Lapse Since Programming 2	Yes
Introductory Programming Language	Yes
Track (Concentration)	Yes
General GPA	Yes
Logic Course	No
Major	No
Faculty	No
Gender	No
Number of Programming Courses Taken	No
Math Courses Taken	No

The independent variables that produced a difference in the Java grades were: time lapsed since Programming 2, the introductory language taken, the track (concentration within the Computer Systems Department) taken, and general GPA (grade point average). The variables that did not produce a difference in the Java grades were: taking a logic course first, major, the particular faculty member that taught the introductory class, gender, number of programming classes taken, and math courses taken.

Time Lapse since Programming 2

Students who took Java the semester following the last introductory programming course had a higher mean average in the Java class than students who waited two or three semesters to take the course. The longer the time lapse, the more the more the mean average declined. See figure 1.

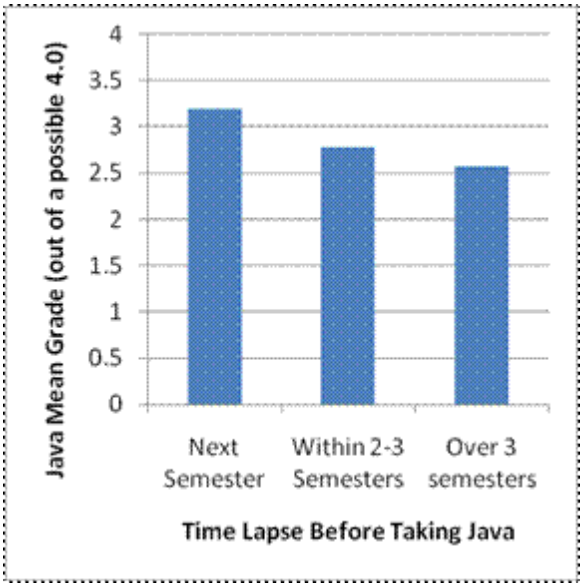

Figure 1: Time lapse between Programming 2 and Java and mean averages in Java

Along these same lines, the longer students put off taking Java after completion of Programming 2, the more likely they were to get below a 2.0 ("D" or "F") in the Java class. Of the students who took Java the following semester after Programming 2, 10% earned a "D" or "F" (under a 2.0 out of a possible 4.0). Twenty percent of students who waited two to three semesters to take Java after Programming 2, received a grade of "D" or "F". Twenty-two percent of students who waited over

three semesters received a grade of "D" or "F". See figure 2.

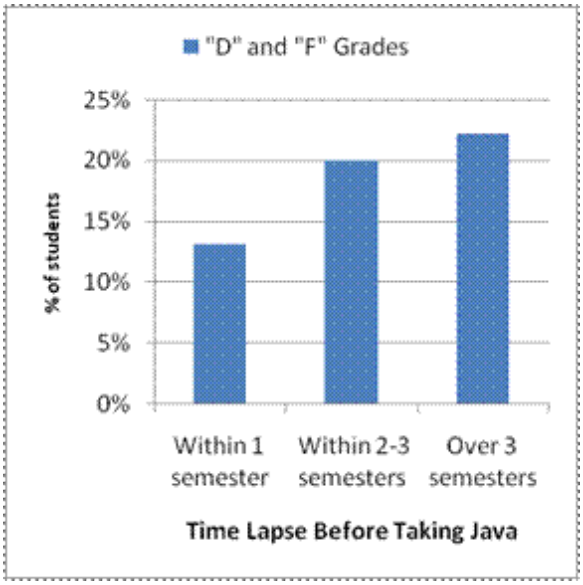

Figure 2: The number of "D" and "F" grades increase when students postpone taking Java.

The statistical validity of these findings was tested using a one-tailed Mann Whitney U Test. There was a significant statistical difference when *next semester* and *within 2-3 semesters* were compared. See table 2.

Table 2

Mean Averages of Java students Grouped by Time Lapse of Next Semester vs. Within 2 or 3 Semesters

	Next Semester	Within 2-3 Semesters
Mean	**3.19**	**2.79**
N	61	75
U = 1884		
Significance= p<.05		

There was a highly significant difference when *next semester* and *over 3 semesters* was compared. See table 3.

Table 3

Mean Averages of Java Students Grouped by Time Lapse of Next Semester vs. Over 3 Semesters

	Next Semester	Over 3 Semesters
Mean	**3.19**	**2.58**
N	61	45
U = 969.5		
Significance= p<.01		

The next semester mean was also compared to the average means for all students who waited over one semester and that result was found very significant. See table 4.

Table 4

Mean Averages of Java Students Grouped by Time Lapse of Next Semester vs. Over 1 Semester

	Next Semester	Over 1 Semester
Mean	**3.19**	**2.71**
N	61	120
U = 2853.5		
Significance= p<.01		

When means for a time lapse of one, two or three semesters were compared to over three semesters that was also found statistically significant. See table 5.

Table 5

Mean Averages of Java Students Grouped by Time Lapse of 1, 2, or 3 Semesters vs. Over 3 Semesters

	1, 2 or 3 Semesters	Over 3 Semesters
Mean	**2.97**	**2.58**
N	136	45
U = 2441.5		
Significance= p<.05		

The only comparison where a statistical significance was *not* found was when *two to three semesters* was compared to *three semesters*.

Introductory Programming Language

It was found that students who took C++ for introductory programming classes were more successful than students who took Visual Basic for introductory programming classes using a one-tailed Mann-Whitney U test.

C++ students in the 2005-2009 group attained an average grade of 2.80 on a 4.0 scale in Java. Visual Basic students in the 2005-2009 group attained a 2.13 grade in Java.

Table 6

Mean Averages of Java Students Grouped by Introductory Programming Language Taken from 2005-2009

	C++	VB
Mean	**2.80**	**2.13**
N	76	28

U = 818

Significance= p<.05

When this data was added to the spring 2010 semester, there was little difference. The C++ average was then 2.78 and the Visual Basic average 2.0. See table 7.

Table 7

Mean Averages of Java Students Grouped by Introductory Programming Language Taken from 2005-2010

	C++	VB
Mean	**2.78**	**2.20**
N	90	33

U = 1189

Significance= p<.05

Track/Major

The Computer Systems Department has five tracks (concentrations) in a particular area. Each student selects one track and completes four courses in that area in addition to taking the other required courses in the curriculum. The two introductory programming courses and the advanced Java course are part of the core required curriculum, not a particular track. The five tracks are programming, web development, networking, systems and database. The database track was added last semester and as a result, was not considered in this research study.

It was found that students in the programming track were most successful in the Java course, followed by networking, web development, undecided, and systems. See figure 3 below which shows average means on a 4.0 scale for the four tracks and students who were undecided.

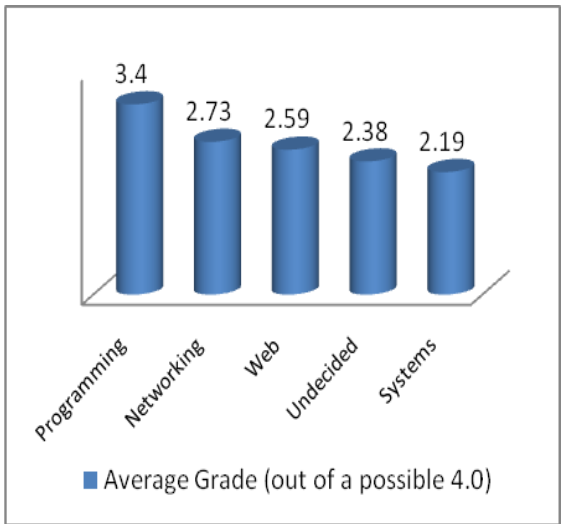

Figure 3: Success in the Java course by track

This difference was found to be highly significant using both a two-tailed Mann-Whitney U Test and a one tailed Mann-Whitney U Test. See tables 8 and 9 below.

Table 8

Comparison of Programming Track vs. ***Not*** *Programming Track – Java Means*

	Programming Track	**Not** Programming Track
Mean	**3.4**	**2.53**
N	45	180

U = 2579

Significance = p<.01

Table 9

Comparison of Programming Track and Other Tracks - Java Means

	Prog.	Net.	Sys.	Web Dev.	Undecided
Mean	**3.4**	**2.73**	**2.19**	**2.62**	**2.38**
N	45	56	29	50	32
U		927.5	298.5	701	445.5
Significance		p<.01	p<.01	p<.05	p<.01

The average mean of the systems track students was then compared to the average mean of all other tracks combined using a one-tailed Mann Whitney U Test. The findings were found significant at p<.01. See table 10.

Table 10

Comparison of Systems vs. Other Tracks – Java Means

	Systems	All Other Tracks
Mean	**2.19**	**2.78**
N	29	196
U = 3622.5		
Significance = p<.01		

Occasionally, students from outside the department take the Java class as an elective. Some of the other majors that have taken this course are nursing, bioscience, applied mathematics, and computer engineering. Also, it is taken infrequently by non-matriculated students who do not have a major. There was no significant difference found between the Computer Systems majors and non-majors.

General GPA

The student's general GPA average in the semester before the student took the Java course was compared to the grade the student earned in the Java class. A highly significant correlation was found between the student's general GPA and the Java grade using the Pearson product moment correlation coefficient (n=225, df=223, r = .52, p<.0005).

Logic Course

In an effort to improve performance in its programming classes, the department changed its requirements a few years ago to include a mandatory programming logic course. This logic course must be taken before the first programming class. No significant statistical difference was found between students who did or did not take the logic class before entering the first programming class.

Faculty

To determine if the particular faculty member teaching the introductory courses affected student success in the advanced Java course, we broke down the Java classes into groups based on the particular instructor that taught the introductory level class. No significant statistical difference was found in the final Java grades based on the faculty member who taught the introductory programming courses.

Gender

Females constituted only 11.60% of the students in the Java classes. Their mean average in the Java class was 2.66 out of a possible 4.0. Males in the Java courses (88.39%) had a mean average score of 2.7 out of a possible 4.0. Thus, no significant difference was found based on gender.

Number of Programming Courses Taken

The college offers a number of additional programming courses that are not required and can be taken as electives. Also, students may take C++ in the introductory courses and Visual Basic as an elective or vice versa. No significant statistical difference was found in the final Java grades for students who took more programming courses than required.

Math Courses Taken

Students are required to take two mathematics courses, Calculus and Methods in Operation Research. These math courses are not prerequisites for the Java course. It was found that there was no significant statistical difference between students who took Calculus before the advanced Java course and students who took calculus after the Java Course.

5. DISCUSSION

Based on the literature review, we expected to find that students who completed the newly required logic class, took Calculus before Java, and completed more programming classes than required would be more successful in advanced Java than students who did not. These

factors, however, were all found to be statistically insignificant for our students.

It is surmised that the logic course may help prepare the students for programming, but not actually increase their programming ability. Anecdotally, instructors in the early introductory classes have stated that it is easier to teach programming to students after completion of the logic class. The instructors found that moving the material covered in the logic course out of the introductory programming course allowed them to devote more time to programming and gave them more time to cover all the required material. Thus, the course still appears to have value and will most likely be maintained in the curriculum.

It appears that the additional programming courses taken by some students did not help them succeed in the advanced Java course. Possibly these additional courses only serve to reinforce and reiterate material already covered. Another explanation might be that students may have difficulty transferring the skills from one language to another. A more accurate and comprehensive exploration of this issue will be undertaken in stage two of this research study. Stage two will use a qualitative approach with in-depth student interviews.

As stated previously, students who completed the required calculus course did not achieve better results in Java. We were somewhat surprised at this finding and recommend further research in this area.

As it appeared in the literature, our study found no significant difference between the performance of men and women. We have too few women entering the field. Those women that do enter, however, are as successful as men.

The fact that some of our students take the introductory programming classes as freshmen or juniors and then do not take the advanced Java class until close to graduation has been mentioned by some faculty as a problem area. This study validates this concern. Programming concepts and theory can be easily forgotten if not reinforced and applied immediately. The department may also have contributed to this problem by not offering the course every semester in the day and evening sessions. This success factor is relatively easy to implement. Students need to be advised to take the Java course immediately after completing Programming 2 and the department has to offer the course every semester, day and evening, with as many sessions as needed.

On the other hand, it may not be the delay itself that causes the later problems in the advanced programming classes. It may be that some students feel insecure with programming itself and thus delay taking the advanced course because of these feelings of insecurity. We plan further investigation in this area using follow-up student interviews.

The results of the study seem to indicate C++ may provide a better foundation for upper level programming in Java. There could, however, be any number of factors to explain this. C++ is closer in syntax to Java and may make the transition to that language easier. On the other hand, it may simply be that the better programmers tend to take C++ instead of VB. This is another area that will be well served by more research of a qualitative nature and student interviews.

It does not seem surprising that students in the programming track would do better in Java than students in the other tracks. Systems students had the worst Java grades. Systems students may have already made the decision to avoid or dislike programming. This brings up the issue of whether or not all information systems students need advanced programming. Are we forcing them to take a course they do not like and do not do well in? Will programming ever be a part of their careers? This topic requires further study outside the realm of this project.

Finally, students that have a better general grade point average do better in Java. Good study skills and habits help a student succeed in any subject. Motivational and psychological factors are important in all academic fields. Students who strive for good grades will want good grades in all their classes. Helping our students to learn and attain good study habits, organizational skills, testing practices, etc. should help students do well in Java as well as their other courses.

6. CONCLUSIONS

Based on the results of this study, students should be strongly encouraged to take Java immediately after completing Programming 2. Programming concepts and theory can be easily forgotten if not reinforced and applied immediately. The department should also do

their part and offer Java in both the spring and fall semesters for day and evening sessions.

The department should consider mandating C++ as a required introductory language and offer Visual Basic as an elective. As mentioned previously, C++ is similar to Java and may make the transition to Java easier. It will also make it easier for the instructor if all students have the same background and entry level skill sets.

This study did not consider whether all information systems students need to take advanced programming. It does suggest that this matter should be researched and discussed. How many programming classes are needed for students who do not intend to become programmers?

This study was limited to only one college and this college may be different than other colleges. The results, therefore, may not be generalizable. Further research at other schools or a consortium of other schools would help to alleviate this limitation.

This study was also limited by its use of final grades as assessment measures. A student's final grade is composed of numerous factors including class participation, objective tests, homework, etc. In this study, we were looking at only one part of this grade — success in programming. It was hard to weed out that one factor from the overall picture. In the future, we plan to give assessment tests in the programming classes to use as comparison measures.

In addition, we would like to enhance the research study by looking at some of the personal and psychological factors that may affect a student's success in the Java class. For this later study, we would like to conduct a survey and perform in-depth interviews with Java students.

7. REFERENCES

Bennedsen, J. & Caspersen, M. E. (2005). An Investigation of Potential Success Factors for an Introductory Model-Driven Programming Course. *Proceedings of the First International Workshop on Computing Education*, Seattle, WA, 155-163.

Bhatnagar, N. (2009). A Study of the Inclusion of Programming Languages in an Undergraduate Information Systems Curriculum. *Information Systems Education Journal,* 7 (84).

Byrne, P. & Lyons, G. (2001). The Effect of Student Attributes on Success in Programming. *Proceedings of ITiCSE 2001*, Canterbury, Kent, ACM Press.

Chen, T., Monge, A. & Simon, B. (2006). Relationship of Early Programming Language to Novice Generated Design. *ACM SIGCSE Bulletin*, 38(1), 495–499.

Dehinbo, J. (2006). Determining Suitable Programming Language for the Bachelor of Technology (IT) Curriculum. *Proceedings of the Information Systems Education Conference 2006*, Dallas, 23.

Guzdial, M. & Soloway, E. (2002). Teaching the Nintendo Generation to Program. *Communications of the ACM*, 45(4), 17-21.

Lahtinen, E., Ala-Mutka, K. & Jarvinen, H. (2005). A Study of the Difficulties of Novice Programmers. *ACM SIGCSE Bulletin*, 37(3), 14-18.

Lewandowski, G., Gutschow, A. McCartney, R. Sanders, K. & Shinners-Kennedy, D. (2005). What Novice Programmers Don't Know. *Proceedings of the First International Workshop on Computing Education Research,* Seattle, WA, 1–12.

Lister, R., Adams, E.S., Fitzgerald, S., Fone, W., Hamer, J., Lindholm, M., Mc Cartney, R., Mostrom, J.E., Sanders, K., Seppala, O., Simon, B. & Thomas, L. (2004). A Multi-National Study of Reading and Tracing Skills in Novice Programmers. *ACM SIGCSE Bulletin*, 36(4), 119-150.

Mannila, L. & de Raadt, M. (2006). An Objective Comparison of Languages for Teaching Introductory Programming. *Proceedings of the 6th Baltic Sea Conference on Computing Education Research*, Uppsala, Sweden. 276, 32-37.

Mannila, L., Peltomaki, M. & Salakoski, T. (2006). What about a Simple Language? Analyzing the Difficulties in Learning to Program. *Computer Science Education,* 16(3), 211-228.

McCracken, M., Kolikant, Y., Almstrum, V., Laxer, G., Diaz, D., Thomas, L., Guzdial, M., Utting, I., Hagan, D. & Wilusz, T. (2001). A Multi-National, Multi-Institutional

Study of Assessment of Programming Skills of First-Year CS Students. Annual Joint Conference Integrating Technology into Computer Science Education. *Working Group Reports from ITiCSE on Innovation and Technology in Computer Science Education*, Canterbury, UK, 125-180.

McIver, L. & Conway, D. (1996). Seven Deadly Sins of Introductory Programming Language Design. *Proceedings of the 1996 International Conference on Software Engineering: Education and Practice* (SE-EP '96), Dunedin, New Zealand, IEEE Computer Society, 309-316.

McKinney, D. & Denton, L.F. (2004). Houston, We have a Problem: There's a Leak in the CS1 Affective Oxygen Tank. *Proceedings of the 35th SIGCSE Technical Symposium on Computer Science Educatio*n, Norfolk, VA, ACM Press, 236-239.

Parker, K., Chao, J., Ottaway, T., & Chang, J. (2006). A Formal Language Selection Process for Introductory Programming Courses. *Journal of Information Technology Education*, 5, 133-151.

Prasad, C. & Li, X. (2004). Teaching Introductory Programming to Information Systems and Computing Majors: Is there a Difference? *Proceedings of the Sixth Conference on Australasian Computing Education*. Dunedin, New Zealand, 57, 261-267.

Rountree, N., Rountree, J. & Robins, A. (2001). Identifying the Danger Zones: Predictors of Success and Failure in a CS1 Course. *Inroads SIGCSE Bulletin*, 34, 121-124.

Rountree, N., Rountree, J. Robins, A. & Hannah, R. (2004). Interacting Factors that Predict Success and Failure in a CS1 Course. *ACM SIGCSE Bulletin*, 36(4), 101-104.

Russell, J., Russell, B., Pollacia, L. & Tastle, W. (2009). A Study of the Programming Languages used in Information Systems and in Computer Science Curricula. *The Proceedings of the Information Systems Education Conference* 2009, Washington DC , 26.

Wiedenbeck, S. (2005). Factors Affecting the Success of Non-Majors in Learning to Program. *Proceedings of the First International Workshop on Computing Education* Research, Seattle, WA, 13-24.

Wilson, B. & Shrock, S. (2001) Contributing to Success in an Introductory Computer Science Course: A Study of Twelve Factors. *ACM SIGCSE Bulletin,* 33(1),184-188.