Improving the Affective Element in Introductory Programming Coursework for the ”Non Programmer” Student

Dr. David M Whittinghill, Purdue University, West Lafayette

Dr. David Whittinghill is an Assistant Professor of Computer Graphics Technology and Computer and Information Technology. Dr. Whittinghill' s research focuses on simulation, gaming and computer programming and how these technologies can more effectively address outstanding issues in health, education, and society in general.

Dr. Whittinghill leads projects in pediatric physical therapy, sustainable energy simulation, phobia treatment, cancer care simulation, and games as a tool for improving educational outcomes. Dr. Whittinghill is the director of GamesTherapy.org.

Prior to joining Purdue he was a senior software engineer in the research industry focused upon the fields of visualization, games, agent-based modeling, digital anti-tampering, robotics, pharmaceuticals, and web development. His primary skills expertise is in computer programming.

Dr. David B Nelson, Purdue University
Mr. K. Andrew R. Richards, Purdue University
Dr. Charles A Calahan

Improving the Affective Element in Introductory Programming Coursework for the "Non Programmer" Student

Abstract

Over a period of several semesters, we examined undergraduate students who were enrolled in an introductory computer-programming course. The goal of the study was to observe the degree to which each student's feelings about the discipline of programming were affected by their experience in this course. The course attempted to encourage a learning environment in which students who were unfamiliar or intimidated by the discipline of programming would be informed that the course is explicitly oriented toward them, rather than toward the more advanced students. The course was designed to defer to the needs of low-skill students such that content progression was slow, thorough, and student centered. Students were surveyed at the beginning of the semester on measures of: self-identified programming skill, years of previous programming experience, and like or dislike of programming. Students were then solicited at the end of semester and surveyed on the degree to which they increased or decreased their enjoyment of programming. As the focus of this approach was oriented toward students with low-positive feelings toward programming, we grouped students into groups of high-positive (HP) and low-positive (LP), and then compared their individual change of attitude toward programming at the end of the semester. We observed that LP students reported greater measures of positive affect toward programming by the end of the semester. These results indicate that approaches to increasing interest in programming education must be accompanied by a supportive, student-centered learning environment that acknowledges the difficulty of the subject matter.

Introduction

Numerous studies have identified and explored barriers to computer programing education from elementary through post-secondary schools. These studies have been motivated in part by consistent calls from government agencies to mediate an apparent shortage of computer programmers or a perceived lack of diversity in STEM fields like programming ([15, 16]). There is some evidence to suggest that a shortage in an economic sense does not exist, but rather the persistence of unfilled posts in programming and IT fields originates more from business hiring practices and a perceived "skill deficit" among degree holders and potential applicants ([2, 5]). Regardless of the veracity of the claims, researchers have been working during the last 15 years to identify and overcome potential barriers to careers and study in computer science. Following recommendations from national reports ([1, 15]), mediation efforts have adroitly focused on K-12 education, measuring potential bottlenecks in the pipeline for programmers ([4, 22]). These efforts have also spurred novel solutions to increase interest and skill in computer programming among students, ranging from content-based solutions to inventive graphical learning tools and storyboard techniques ([4, 10]). The field is experiencing an increased emphasis

on programming concepts' education in lieu of an exclusive focus on syntax language learning ([12, 17]).

These innovative approaches and techniques have also extended into research and practice at the post-secondary level. Critiques of the predominant "objects-first" approach encourage a rethinking of the paradigmatic structure of many introductory college programming courses ([8, 13]). Utilization of software packages provides students with alternative structures to learn and manipulate programming environments outside of simple code syntax ([9, 11, 20]).

Extension of these tools to higher education targets a higher-stakes environment for many students. The abbreviated calendar of classes and the significant weight of university grades often creates a narrow window into which students can become successful in a course ([3, 21]). For programming in particular, early failure to understand concepts in college courses undermines students' confidence and competence ([6]). Many of the learning techniques in the literature target affective resistance to programming as a way to increase retention of these students. Approaches that emphasize team-based learning and peer-projects posit that a sense of belonging ([14]) improved self-sufficiency and competence in higher-order thinking skills ([23]) will strengthen retention in programming.

There is a tenuous link in the literature between novel learning approaches and students' affective attitudes towards programming. Most explorations of effective programming-education software are tied to academic success or performance relative to instructor criteria ([9, 19]). Work on alternative course-based structures in higher education has more directly incorporated programming attitudes and aptitudes into research ([14, 23]). However the scope of most affective studies remains within the confines of a particular course, rather than exploring students' attitudes towards general programming and likelihood of continued pursuit in the field.

In this paper, we suggest that significant value lies in an examination of the learning environment of college programming courses, particularly introductory courses that are populated with novice computer programmers. Studies in both K-12 and post-secondary programming education hint that an inclusive and supportive learning environment can improve student motivation and confidence more than any particular pedagogical technique of tool ([10, 18, 20]). Studies in motivation and self-determination support the value of autonomy, competence and relatedness as integral to academic success ([7]).

Class Environment and Student Population

Our study was conducted over six academic semesters of an introductory computer-programming course at a large Midwestern university from the spring 2011 semester through fall 2013. The class is a required course for majors in the department, and is the only required programming course they will take. Second-year students are the most dominant demographic group in the course. Since the major caters to many different vocations and interests, students in the class may have had no prior programing experience. Similarly, many have expressed anxiety about their mathematical and

programming abilities. The curriculum of the course has targeted explicitly these beginning programmers. The instructor emphasizes to students with extensive programing experience that the course will not advance their knowledge of the field but will reinforce it. Thus, the course can accurately be described as one that focuses on the success of Low Experience/High Anxiety students.

Additionally, the sixth and most recent semester of the course was redesigned in tandem with a campus initiative to increase student-centeredness and higher-order thinking skill acquisition in foundational courses. Accordingly, the fall 2013 semester incorporated team-based learning into classroom activities. Students worked with their peers to analyze coding scenarios that were often beyond the knowledge and experience of a novice programmer. The teams were tasked with exploring possible solutions and charting their progress on large sheets of paper. The instructor moved among the teams at their various tables. Students were required to explain their reasoning behind certain choices, and any team member could be expected to discuss the group progress with the instructor. This allowed the students with limited programming experience to explore coding in a safe environment and receive feedback regularly and consistently, while maintaining an understanding of the programming techniques that the group chose. Methods and material from the coding challenges were then assessed through in-class exams and weekly independent labs.

Methodology

During six semesters, 203 students in the introductory programming course completed two surveys that solicited their attitude towards programming. Surveys conducted in the first weeks of the semester asked students to rate their programming skills, quantify the number of years of programming experience, indicate whether they liked or disliked programming, and identify whether they would have enrolled in the course if it were not required of them. At the conclusion of the course, students were asked to reflect on the contribution of the course to their enjoyment of programming. We used this data to create a pre and post-score, where like/dislike of programming was measured in the beginning and end of the course.

We used this data to test four research questions about attitudes toward programming:

1) Can an inclusive, supportive environment that is catered to the non-programmer lead to improved attitudes about programming?
2) Can students with low-positive feelings (LP) increase their confidence in programming?
3) Does prior experience with programming influence the degree of attitudinal change?
4) Do specific pedagogical techniques and practices in a redesigned course influence attitudinal change?

Analysis

Our paper proceeds under the assumption that the introductory course represented an inclusive and supportive environment. We did not use any external evaluative protocols or measures to reach this determination. Rather, as described in the introduction, the course and its assignments were designed to benefit the students with high anxiety towards programming. The course involved regular in-class group work, providing students an opportunity to practice various coding challenges with immediate feedback from their peers and the instructor. Practice, time on task, and mistakes were all encouraged, with progress being charted throughout the semester. Students presented their work to one another, and each group could be responsible for discussing their conclusions with the class. This approach both integrated students who might not otherwise participate, and allowed them opportunity to explore concepts and misunderstandings in safe environments with their peers.

The beginning of semester survey questions were as follows:

1. How would you rate your programming skills?
    a. 5-point Likert scale
    b. Have never programmed before; Below average; Average; Above Average; Expert
2. How many years of programming experience do you have?
    a. 6-point Likert scale
    b. None; < 1 year; 1-2 years; 2-3 years; 3-4 years; 4 or more years
3. How much do you like/dislike programming?
    a. 5-point Likert scale
    b. Dislike Extremely; Dislike Very Much; Neither Like nor Dislike; Like Very Much; Like Extremely
4. Do you think that having some programming skills can help you with your particular career goals?
    a. Yes or No
5. Would you take this course if it was not required?
    a. Yes or No
6. What job are you hoping your job in XXX* will prepare you to do?

The end of semester survey questions were as follows:

1. Given that XXX is an introductory programming class, I felt the difficulty of the material was:
    a. 5-point Likert scale
    b. Far too easy; A little too easy; Just right; A little too hard; Far too hard
2. I felt the difficulty ***progression*** from easy to hard was:
    a. 5-point Likert scale
    b. Far too slow; A little too slow; Just about right; A little too fast; Far too fast
3. I am a better programmer because of this class
    a. 5-point Likert scale
    b. Very much disagree; Somewhat disagree; Neither agree nor disagree; Somewhat agree; Very much agree

4. Based on my experiences in this class, I now enjoy programming:
    a. I dislike programming a lot
    b. I dislike programming a little
    c. I neither like nor dislike programming
    d. I like programming a little
    e. I like programming a lot
5. For my individual learning preference, the ideal distribution between in-class activities versus in-class lectures by the professor would be:
    a. (For 100% in-class ***activities***, select 10, for 100% in-class ***lectures***, select 0)
    b. [Slider widget from 0 to 10]
6. I would like to take more programming classes like this one
    a. 5-point Likert
    b. Strongly disagree; Somewhat disagree; Neither agree nor disagree; Somewhat agree; Strongly agree

**The designation XXX was used to obscure identifying information for the review process.*

With this central assumption guiding the research, we undertook four distinct, but related analyses to answer our hypotheses. First, we examined the possible effects of new pedagogical techniques in the redesigned course (fall 2013) to determine whether the responses gathered should be evaluated distinctly from the previous five. Independent-samples t-tests were conducted on student responses to the summative question, "Based on my experiences in this class, I now enjoy programming." While students in the fall 2013 class reported slightly greater degrees of post-course enjoyment, the differences were not enough to be sufficient. Despite the marked focus on team-based activities in fall of 2013 in lieu of the lecture format that had characterized coursework in previous semesters, student attitudes towards programming and the changes in those attitudes remain nearly identical across all classes.

After we determined that we could analyze data from all semesters in aggregate, we examined the temporal element of student attitudes towards programming. We compared student responses to the initial attitudinal question, "How much do you like/dislike programming" the summative question, "Based on my experiences in this class, I now enjoy programming." Using a paired-sample t-test to measure if participants' attitudes became more positive from time one to time two. The test statistic was significant ($t(201)=7.98$, $p=<.001$), and is associated with a medium-effect size ($d=0.67$). Students' attitudes toward programming were more positive at the end of each semester than they were in the beginning.

As an isolated analysis, the change in student attitudes is not particularly surprising. In most programs of study, and in sufficiently large sample sizes, one can generally find that exposure to and familiarity with a subject improves a student's attitude, especially if the student has not previously encountered the topic or discipline. However, when we examined these general conclusions along with other variables, we can see that the

particular emphasis this course places on students with high-dislike for programming results in significant affective gains.

Since the multiple changes in activities and lessons in the re-designed semester did not foster a significantly different attitudinal experience for the students enrolled, we are also working on the assumption that the activities and group-work themselves did not create an improvement in attitude. Rather, the general environment of the course was relatively consistent throughout three years with the same instructor, and was sufficiently inclusive and supportive to improve attitudes for all students. In order to trace the relative effectiveness of the course environment upon those students with high-dislike for programming, we incorporated a student's prior experience with programming and their self-estimate of their programming skill. These measures were both taken at the beginning of each semester.

Our hypothesis initially supposed that students' prior experience with programming and their subjective ratings of programming skill would significantly predict their programming enjoyment above and beyond their initial enjoyment levels. To test this hypothesis, we ran a hierarchical linear regression analysis (See Table 1). This analysis allowed us to control for the variance accounted by the pre-semester enjoyment variable in evaluating the impact of years of coding experience and rating of programming skill on the end of semester enjoyment rating. In the first step, only the pre-score is entered as predictor. We find that it is a significant predictor, ($F(1,201)=19.92$, $p<.001$). The effect size is small ($R^2=.09$). In the second step, our goal was to examine whether or not the past experience and personal rating variables are useful in predicting post-enjoyment scores above and beyond that which was accounted for by the pre-score. The omnibus F-Test was significant ($F(3,199)=7.76$, $p<.001$), and associated with a small effect size ($R^2=.11$). However, the test for change in R^2 was not significant ($F(2,199)=1.62$, $p=.202$). Indicating that the additional variance accounted for by including the new variables does not result in a significant increase in the amount of variance explained. Examination of the tests for individual coefficients confirms that the pre-semester enjoyment score is still significant ($\beta=.33$, $p<.001$), but neither subjective rating of programming skill ($\beta=-.09$, $p=.448$), nor years of prior programming experience ($\beta=-.04$, $p=.747$), were significant. Based on this analysis, we concluded that neither variable predicts the summative response beyond the variance that is already accounted for in the pre-survey "like/dislike" question.

Table 1.

Summary of Hierarchical Multiple Regression Analysis

Predictor Variables	Step One	Step Two
Pre-semester enjoyment	.300**	.334**
	(4.46)	(4.79)
Years of programming experience		$-.038^{NS}$
		(-.323)
Perceived programming skill		$-.091^{NS}$
		(-.760)

R^2	.090	.105
Adjusted R^2	.086	.091
N	203	203

Note: The dependent variable was post-semester programming enjoyment. Standardized regression coefficients are reported and *t*-values are in parentheses. *Significant at α=.05, **Significant at the α=.01 level, ***Significant at the α=.001 level, ^{NS} Not significant.

Through these combined analyses, we were able to suggest with some certainty that the course helped improve attitudes towards programming for all students, regardless of each student's background and comfort with programming. The specific modifications made during the course redesign also appear to have a negligible effect, though the sample size is small, and analyses of future semester data could complicate that picture further, especially as the instructor becomes more familiar with and confident about the new group-work and assignments.

Nonetheless, our analyses did not provide a satisfactory explanation for the assumption inherent in the initial research question. While all students seem to increase their positive attitudes toward programming, were the students, with an apprehension toward programming, reaping any particular benefits of a course tailored specifically to them?

In the pre-survey, students were asked to indicate on a binary level whether they would have taken the course if it were not required for the plan of study (TakeCourse). This provides a useable proxy for students with high anxiety/high-dislike towards programming. We ran a 2x2 (Time x TakeCourse) Mixed ANOVA (See Table 2), examining both the change in attitude and desire to take the course if not required. Results indicate a significant main effect for change in attitude over time, ($F(1,201)=65.72$, $p<.001$), which is associated with a large effect size (partial-$\eta^2=.246$). There was also a significant main effect for TakeCourse ($F(1,201)=8.98$, $p=.003$), which was associated with a small effect size (partial-η2=.043). This indicates that the students who would have taken the course if it had not been required report greater enjoyment at both pre- and post-semester. Both main effects were qualified by a significant Time x TakeCourse interaction effect ($F(1,201)=4.68$, $p=.032$), which was associated with a small effect size (partial-η2=.023). As demonstrated in the means plot in Figure 1, students who report that they would not have taken the class if it had not been required begin the semester with lower enjoyment scores, but catch up to their peers who would have taken the course by the end of the semester. On the means plot appears as if there are differences between the two groups of students' enjoyment scores on the pre-assessment, but not on the post-assessment. In order to test for this, we used an independent-samples *t*-test to examine simple effects. Results confirm that there was a statistically significant difference between the students who would and would not have taken the class if it were not required on the pre-test ($t(201)=4.65$, $p<.001$). This significant difference was associated with a medium effect size ($d=.74$). While there was a significant difference at the beginning of the semester, differences between the groups were not significant at the end of the semester. This provides evidence to support the assertion that students who would *not* have taken the course if not required start with a

greater attitudinal dislike for programming than their more enthusiastic peers, but the gap closes almost completely by the end of the course.

Table 2.

ANOVA table for the for enjoyment of programming at pre- and post-semester based on whether or not the students would have taken the course if it had not been required

Variable Time	TakeCourse		ANOVA Statistics			
	Yes M(SD)	No M(SD)	Factor	F	P	Partial-η^2
Enjoyment			Time**	65.72	<.001	.246
Pre	3.13(.69)	2.60(.77)	TakeCourse*	8.98	.003	.043
Post	3.65(1.06)	3.49(1.03)	Interaction*	4.68	.032	.023

Note: *p<.05, **p<.01, TakeCourse=whether or not the student would have taken the course if it had not been required.

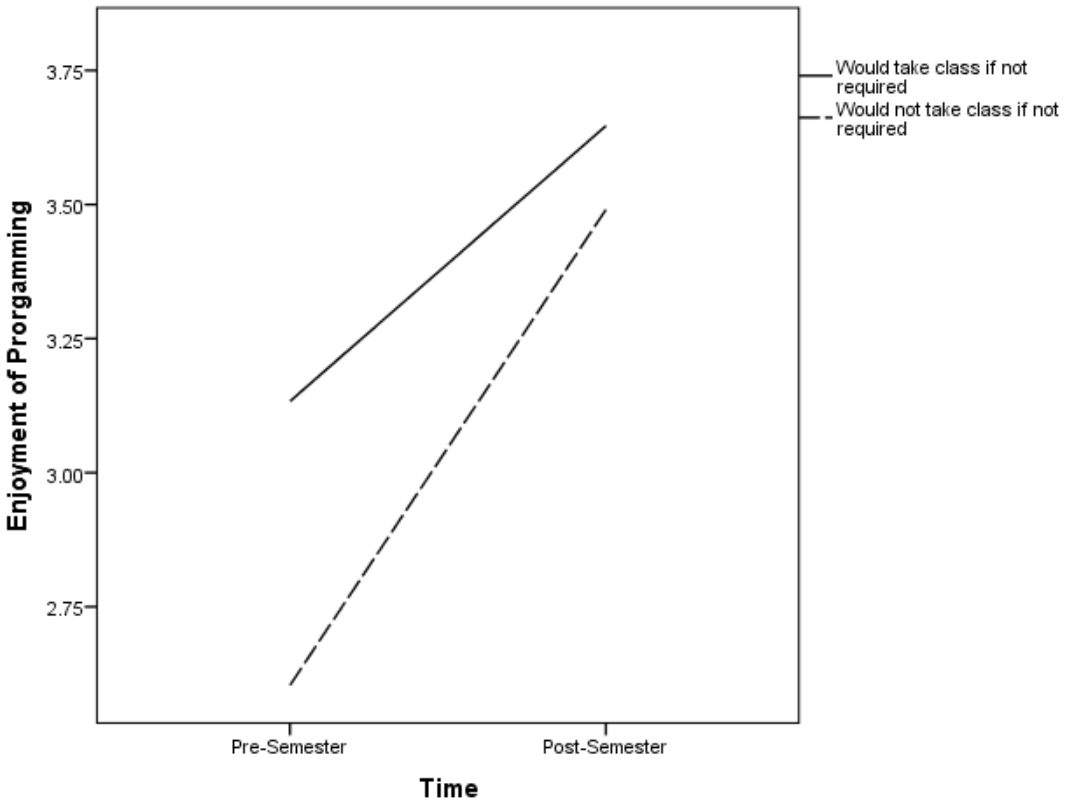

Figure 1. Means plot displaying enjoyment of programming at pre- and post-semester based on whether or not students would have taken the class if it were not required

Conclusion and Discussion

Students in the introductory programming course above reported significant positive attitude changes regarding programming and their respective skill in it. The course was structured to emphasize the process of coding in a forgiving learning environment. In

lieu of memorization of coding syntax or lecture on specific characteristics of the coding language, the course included group learning activities and collective practice at coding challenges. These measures provided students who might otherwise be intimidated or daunted by the prospect of computer programming to improve their affect and confidence for programming tasks. In an initial survey, students indicated their programming skill, quantity of programming experience, and attitude towards programming. A second survey once again measured the increase in their appreciation for programing. Students who initially conveyed anxiety or low-positive (LP) feelings toward programming eventually matched their high-positive (HP) peers' feelings by the end of the course. Variables such as years of programming experience or confidence were not significant factors in the positive change in attitude. Despite the inclusion of an intentional course redesign in the latest semester (fall 2013), the degree and significance of positive change was similar across all six courses. This suggests that while specific techniques in a course may have fostered different learning, the inclusive environment that the instructor created may have greater responsibility for student attitude shifts than any team-based learning activities.

It should be noted that as a result of these findings, changes in the way this course will be taught in future semesters are being enacted. Though the findings are encouraging, improvements can always be made. Most particularly students have advocated for less passive lecture time and more time to work hands-on with coding problems. To address this, the course has been radically re-worked to accommodate this recommendation. Future studies will describe the effects of this change. Also, students reported enjoying the opportunity to work creatively on independent projects. This aspect of the course is being strengthened and broadened so that creativity can be a part of regular assignments and not just the large projects. Future studies are planned to quantitatively assess the impact of these new course evolutions.

Bibliography

1. American Association of University Women, Educational Foundation (2000). Tech-savvy: Educating girls in the new computer age.

2. Bailey, J., & Stefaniak G., IT skills portfolio research in SIGCPR proceedings: Analaysis, Synthesis, and Proposals. *ACM SIGCPR Conference Proceedings.*

3. Barefoot, B.O., Garnder, J.N., Cutright, M., Morris, L.V., Schroeder, C. C., Schwartz, S. W., Siegel, M. J., Swing R. L., (2005). Achieving and sustaining institutional excellence for the first year of college. Jossey-Bass, San Fransisco, CA.

4. Bruckman, A., Jenson, C., and DeBonte, A. (2002). Gender and Programming Achievement in a CSCL Environment. *Proceedings CSCL*, 119-227.

5. Cappelli, P. (2000). Is there a shortage of Information Technology workers? A Report to McKinsey and Company.

6. Chalk, P., Boyle, T., Pickard, P., Bradley, C., Jones, R., Fisher, K. (2003). Improving pass rates in introductory programming. Proceedings of LTSN-ICS 2003, Galway 2003.

7. Deci, E. L. (2009). Large-scale school reform as viewed from the self-determination theory perspective. *Theory and Research in Education,* 7, 244-252.

8. Hu, C. (2004). Rethinking of Teaching Objects-First. *Education & Information Technologies*, 9(3), 209-218.

9. Kelleher, C., & Pausch, R. (2005). Lowering the barriers to programming: A taxonomy of programming environments and languages for novice programmers. *ACM Computing Surveys,* 37(2), 83-137

10. Kelleher, C., & Pausch, R. (2007). Using storytelling to motivate programming. *Communications Of The ACM*, 50(7), 58-64.

11. Li, L., Juarez, J., Yang, Y. (2012). Programming concept visualization using flash animation. *ASEE Proceedings*.

12. Maloney, J., Peppler, K., Kafai, Y., Resnick, M., Rusk, N. (2008). Programming by choice: urban youth learning programming with scratch." *ACM SIGCSE Bulletin* 40(1), 367-371.

13. Mccracken, M., Almstrum, V., Diaz, D., Guzdial, M., Hagan, D., Kolikant, Y. B., Laxer, C., Thomas, L., Utting, I., Wilusz, T. (2005). A multi-national, multi-institutional study of assessment of programming skills of first-year CS students. *ACM SIGCSE Bulletin,* 33(4), 125-180

14. McKinney, D., Denton, L. (2005). Affective assessment of team skills in agile CS1 labs: the good, the bad, and the ugly. *ACM SIGCSE Bulletin*, 37(1).

15. National Science Foundation. (2000). Land of plenty: Diversity as America's competitive edge in science, engineering and technology. Washington, DC: US Congress
16. Office of Technology Policy. (1997). America's New Deficit: The Shortage of Information Technology Workers. Washington, DC: U.S. Department of Commerce.
17. Powers, K., Ecott, S., Hirshfield, L. M (2007). Through the looking glass: teaching CS0 with Alice. *ACM SIGSCE Bulletin*, 39(1), 213-217.

18. Scaife, M. & Taylor, J. (1991). Graduated learning environments for developing computational concepts in 7-11 year old children. *Journal of Artificial Intelligence in Education*, 2, 31-41.

19. Scott, A. (2010). *Using flowcharts, code and animation for improved comprehension and ability in novice programming* (Doctoral thesis, University of Glamorgan, United Kingdom). Retrieved from http://dspace1.isd.glam.ac.uk/dspace/bitstream/10265/460/3/Dr%20Andrew%20Scott%20-%20PhD%20Thesis.pdf.txt

20. Swain, N. (2013). RAPTOR – A vehicle to engage logical thinking. *ASEE Proceedings*.

21. Tinto, V. (2012). Moving from theory to action: A model of institutional action for student success. In A. Seidman, *College student retention: Formula for success* (pp.251-266). Lanham, UK: Rowmn & Littlefield.

22. Tucker, A., Deek, F., Jones, J., McCowan, D., Stephenson, C., & Verno, A. (2003). A model curriculum for K-12 computer science: Final report of the ACM K-12 task force curriculum committee. *ACM, New York*

23. Williams, L., Wiebe, E., & Yang, K. (2002). In support of pair programming in the introductory computer science course. *Computer Science Education*, 12(3), 197-21.