Improving First-year Success and Retention through Interest-Based CS0 Courses

Michael Haungs, Christopher Clark, John Clements, David Janzen
Computer Science
California Polytechnic
1 Grand Ave
San Luis Obispo, CA 93407
{mhaungs,cmclark,clements,djanzen}@calpoly.edu

ABSTRACT

Many computer science programs suffer from low student retention rates. At Cal Poly San Luis Obispo, academic performance and retention rates among first-year computer science students are among the lowest on campus.

In order to remedy this, we have developed a new CS0 course featuring different "tracks" that students can choose from (e.g. robotics, gaming, music, mobile apps). This allows students to learn the basics of programming, teamwork, and college-level study in a domain that is of personal interest. In addition, the course relies on classic Project-based Learning (PBL) approaches as well as a focus on both academic and non-academic factors shown to increase student retention.

Initial assessment demonstrates positive results in the form of increased academic performance in post CS0 courses and student retention.

Categories and Subject Descriptors

K.3.2 [**Computers and Education**]: Computer and Information Science Education—*Computer Science Education*

General Terms

Experimental

Keywords

CS0, CS1, Cornerstone, motivation, self-efficacy

1. INTRODUCTION

The ubiquity of computing in modern society is no longer in doubt. Communication, transportation, education, business, medicine, and entertainment are all, to a certain extent, applications of computing systems. However, despite the rise of computers in our everyday lives, the number of students, and in particular females and minorities, studying computer science as undergraduates has not kept pace with our society's need to train computer engineers and scientists. Further compounding this issue is the fact that many computer science programs suffer from low student retention rates[5]. At Cal Poly, academic performance and retention rates among first-year computer science students are among the lowest on campus. These problems forced us to take a serious look at the goals of our introductory curriculum, and ways in which it could be dramatically reshaped.

1.1 Non-Academic Factors

Typically, the main focus of first-year freshman computer science courses is to build academic proficiency in introductory material. While academic success is correlated to increased student retention, certain non-academic factors also strongly affect it. Examples of non-academic factors may include but are not limited to student confidence, program culture, time management, etc.

Evidence of the effects of non-academic factors can be found in a recent study by ACT[1], that reported a strong correlation between non-academic factors and student success and retention [12]. Their recommendations for improving student retention are complementary with cornerstone, project-based learning initiatives prevalent in the literature and are aligned with our own observations.[2]

Carnegie Mellon University (CMU) greatly increased their enrollment and retention of women in CS [8] by improving several non-academic factors. In particular, they changed the culture of the program, placed it in a social context, closed the experience gap, dealt with student confidence issues, and emphasized the importance of good teaching and faculty-student relationships.

Also of significance are the achievements made at Harvey Mudd College, where a combination of a breadth-first approach to the CS1 course, interaction with computing professionals, and research experiences were used as a means to increase participation in CS [1]. Notably, enrollment of women in CS at Harvey Mudd rose from 11% to 50% from the years 2003 to 2007.

Working in groups has also shown to improve performance. Observations by Williams [15] show increases in academic performance when Pair Programming is used in a CS1 course.

SIGCSE'12, February 29–March 3, 2012, Raleigh, North Carolina, USA.

[1] ACT is an independent, non-for-profit organization specializing in education and workforce development improvement. Historically, ACT stood for "American College Testing".

[2] In the rest of this paper, we use the term *non-academic factors* as defined in ACT's report.

Project Based Learning (PBL) has also been shown as a valid technique for improving learning and academic success [2].

Like CMU and Harvey Mudd, we chose an integrated approach to our CS retention issues where in addition to building a firm grasp of elementary programming concepts, including software testing and development methods, we aim to improve the following non-academic factors: *a student's academic-related skills, academic self-confidence, achievement motivation, and social support and involvement.*

1.2 Interest-Based Tracks

In addition to these proven measures, we embarked upon an experiment in the form of interest-based "tracks."

For the last several years, the CS department has taught an elective Gaming-focused course [10], where students both inside and outside of Computer Science work in teams to build games of their own design. Students in these classes reported that nearly all students characterized computer science as collaborative, multi-disciplinary, and creative.

Building on this success, we decided to expand this to all of our first-year students. Bearing in mind that not all students would be interested in gaming, though, we decided to offer four parallel tracks, each one focused on a different domain. Each of these tracks focuses student attention on applications of Computer Science outside the traditional CS "box," illustrating the interdisciplinary nature of the major and allowing them to work with others outside of the class.

In our first iteration, these tracks included Mobile Applications, Robotics, Gaming, and Music. Each student, upon enrolling, was required to rank these tracks according to their interest level. Students were enrolled in the classes based on these rankings, thus allowing them to self-select their learning communities. The resulting classes feature students that predominantly have higher interest in the course's topic. The authors are not aware of any other computer science program that offers this choice to its first-year students.

In the next section, we describe the course framework common to all versions of the course, regardless of theme. Following this framework, we describe the first four versions of the course offered. Next, we highlight how the organization of the course helps us focus on certain non-academic factors shown to be strongly correlated to improved retention. We then provide our initial assessment of the success of the course. Last, we discuss plans we have for future course improvement.

2. CSC 123 CURRICULUM

Historically, Cal Poly's computer science department has taught an introductory course sequence that is spread across three quarters in the first year of study. Named "Fundamentals of Computer Science" (CSC 101, CSC 102 and CSC 103), the 3 courses cover basics of computer programming in procedural and object-oriented styles, with students working primarily in C and Java. The new course, CSC 123, is taken before CSC 101 and hence shifts the existing sequence back one quarter.

2.1 Common CSC 123 Framework

The tracks all started with a common set of goals, as follows:

Equal participation All course versions will ensure there is balanced participation in the course for students of all programming backgrounds. This will be accomplished by picking projects that are familiar to almost all students (e.g. game design or music), but don't have obvious solutions, allow for creative solutions, and in which prior programming experience is not a significant advantage to student success in the course.

Basic Learning Outcomes All course versions will ensure that students completing the course will have learned certain fundamentals of computer programming, e.g. datatypes, loops, etc.

Software Development Process All course versions will require students to complete a large project. In doing so, students will follow a software development process that includes iterative design, implementation, and testing.

Team-based Learning All course versions will have students working in teams to complete their projects. Team sizes will range from 4-6 students. To provide students with initial guidance on how to function within a team, a short lecture and workshop will be conducted.

Grading Scheme All course versions will use the same set of deliverables and grading rubric on project demonstrations, presentations, reports, and exams. For exams, half of the exam questions will be common to all course versions (see Figure 1 for an example), while the other half will be specific to the application theme.

Student Mentoring All course versions will use a formal student mentoring system in which outside of lecture and lab hours, upper year students will lead workshops on programming within application themes. To improve culture within the program (a known problem for retention as stated by Crenshaw [6]), mentors will be trained to maximize participation from all students and help foster a working community.

Applications with External Clients All course versions will have an external client, typically outside the discipline of computing, who will provide high level direction on the project in terms of its use in the respective discipline.

2.2 Specific CSC 123 Themes

The four initial themes of CSC 123 offered during the Fall quarter, 2010, are given below. As described in the previous section, the different versions of the course have common learning objectives, structure, and desired outcomes. The application themes were used to motivate students. For example, the Computing in Robotics theme taught students about robotics and their use in marine biology. In programming and preparing robots for real ocean deployments and working alongside biologists, engineering students were exposed to situations that they would not otherwise experience in CS. Combined with the fact that application themes were

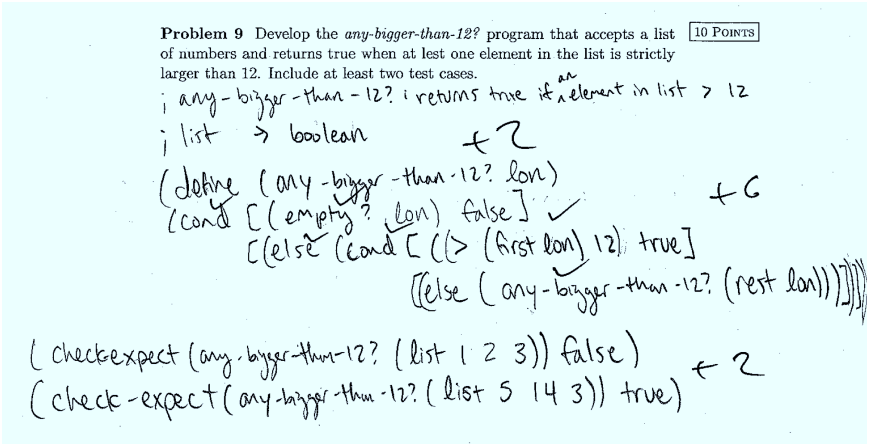

Figure 1: A question and student answer taken from the final exam from the music section of CSC 123. Students in that section programmed in Racket.

preselected by students, it is hoped that the students' motivation toward this course and the field of computing itself will be dramatically increased. Here are the class descriptions of the four different themes given to the students:

Theme 1: Mobile Computing Explore computing and software development by creating apps for Android-based smart phones. Students will work in teams to imagine, design, prototype, build, test, and launch apps for the Android market. Students will gain hands-on experience with the languages and tools needed to build Android apps, and they will test their apps on Android phones.

Theme 2: Computing in Robotics Explore computing and software development by designing a controller and mission planner for an autonomous underwater robot. Students will learn background on underwater robotics and its application to biology, oceanography and marine archaeology. After learning basic underwater robot control, students will deploy robots in Avila Bay California from our lab on the pier. Students will then work in teams to imagine, design, prototype, build, and test their own robot controller and mission planner via real ocean deployments.

Theme 3: Computing in Gaming Explore computing and software development by creating a full featured web-based computer game. This course includes an overview of the development process of games and an introduction to gaming fundamentals: logic, story and game play. The course will focus on design, teamwork, and using an iterative development process. Students will learn basic programming skills in order to develop a simple 2D game using a scripting language (such as Actionscript).

Theme 4: Computing in Music Students will write programs that generate music. Early compositions will focus on the elements of western harmony. Later ones will allow students to explore timbre and digital sound processing. Students will work collaboratively in teams to design, develop, and test their programs and compositions. The course will focus on design, teamwork, and the use of an iterative development process. The course is intended to be an enjoyable introduction to both computer science and elementary composition.

2.3 Stressing Non-Academic Factors in CSC 123

As mentioned in Section 2.1, we have common, basic learning outcomes in all themes of CSC 123. The goal of these outcomes are to increase the students ability to succeed in future CS courses by giving them a firm foundation in basic programming fundamentals. One key aspect to building up student academic skill is the fact that we do not assume any prior programming experience (clearly stated in the course description and syllabus) which helps us to:

1. provide students new to CS a non-threatening, confidence building environment to learn the basics,
2. give students who have learned poor programming habits an opportunity to correct them,
3. and allow students with prior experience an opportunity to practice their knowledge in order to gain mastery.[3]

While academic proficiency is very important to overall student success, a recent meta-study, [12], found that certain non-academic factors are key to student success as well. The factors we considered and built into our course are the ones that are most strongly related to student success. They are academic-related skills, academic self-confidence, social support, and achievement motivation. We describe each and the role the play in our course below.

2.3.1 Academic-Related Skills

Academic-related skills refer to the ability a student has to form good study habits, manage their time, take good notes, and use academic resources, such as the campus library and computer labs. We develop these skills in our students in three ways. First, we provide access to upper year student mentors who have developed successful study habits to meet

[3]A common technique used in the Kumon method of teaching.

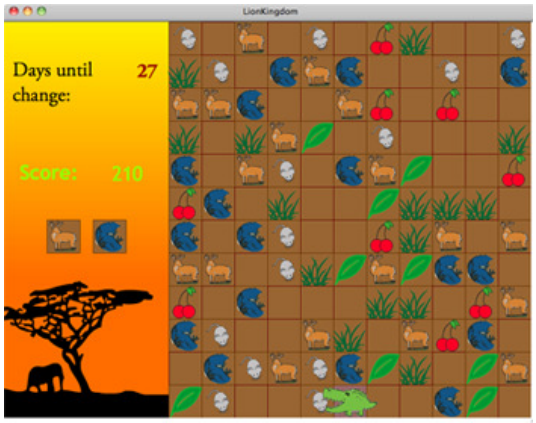

Figure 2: Lion Kingdom, an original game programmed and designed by CSC123 students.

with CSC123 students regularly and help them prepare for upcoming milestones and exams. [13] and [4] demonstrate how effective this type of mentoring can be. Second, we expose our students to a wide variety of assessment techniques typically found in college courses to help them make the transition from high school forms of assessment providing them with the experiences they need to better understand how to acclimate to college academic life. In [9], they found that difficulty adjusting to college life was a factor in poor retention and that activities that smooth this transition are beneficial. In lecture, we discuss good study habits [14], how to plan their time for upcoming class deliverables, and highlight resources available to them.

2.3.2 *Academic Self-Confidence*

Academic self-confidence refers to how successful a student feels they are (or can be) in an academic environment. Students with a low-level of confidence tend to procrastinate or avoid academically challenging activities. In the worst case, it can lead to changing majors, academic dishonesty, or dropping out of college [9, 3]. We attempt a number of ways to boost academic self-confidence. First, we chose topics that most students are already familiar with regardless of their academic background. Contrast the types of discussion that can occur on the first day in a class on introductory Java versus a course on video games. While students with no prior programming experience would feel intimidated in the former, they are quite able to participate in the latter. Figure 2 shows "Lion Kingdom", a game created by a group of students where 4 of the 5 team members had no programming experience.

Our courses include hands-on labs that build student programming skills and confidence as they incrementally master topics within the course domain. Students use both pedagogical and professional development environments. For instance, students in the music version use DrRacket, and students in the mobile version use App Inventor for Android, Eclipse with Java and the Android SDK, and WebIDE [7], an NSF-sponsored lab environment created by two of the authors.

Our courses also stress learning a software development process and applying software best practices such as test-driven development (e.g, [11]). This gives students the tools

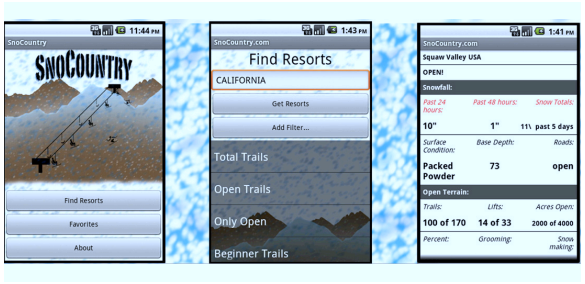

Figure 3: An Android application for skiers complete with instant, online updating of ski conditions.

they need to tackle the type of challenging programming activities they will encounter in later courses. Often, having a clear starting point to a project is all a student needs to feel confident about completing the project.[4] Figure 3 shows a final project done in the mobile development section of CSC 123. While, initially the task to create such an application seemed daunting, the students were carefully led through a process with intermediate milestones to tackle the problem in a productive manner. The first project milestone involved creating a vision and scope, feature list, and user interface mockups for the proposed app. Each student was provided an Android phone, allowing them to explore apps in the Android Market. As a result, every student could contribute to the project idea from the very beginning, regardless of prior programming experience. Subsequent milestones involved development status presentations, testing, product demos, and reflections on the software development experience.

2.3.3 *Social Support*

For the purpose of student retention, it is important that students feel connected to their institution, faculty, peers, and campus activities [12]. To help foster a feeling of connectedness, we have students work in teams, provide upper-level student mentoring, require the students to attend relevant technical talks on campus, and arrange for a wide variety of student clubs to come talk the students. The benefits of student team support are well-documented in the literature.

2.3.4 *Achievement Motivation*

Achievement motivation describes the level at which a student wants to be successful in the class and major. In lecture in CSC 123, we spend time providing a"big picture" view of computer science and how it will allow them to work in a wide variety of industries and make real-world contributions through multi-disciplinary collaborations. CSC123 was constructed to reflect this view. Students are able to work with peers on a substantial quarter-long project in a context they self-selected. For the robotics students, they were able to launch in actual Autonomous Underwater Vehicle (AUV), Figure 4, and have it navigate to predetermined waypoints.

At the end of the course, CSC 123 students present their work to both students in their section and students from other CSC123 sections. We found this activity builds camaraderie within sections and a healthy competition between sections.

[4]To quote Mary Poppins, "Once begun, half done."

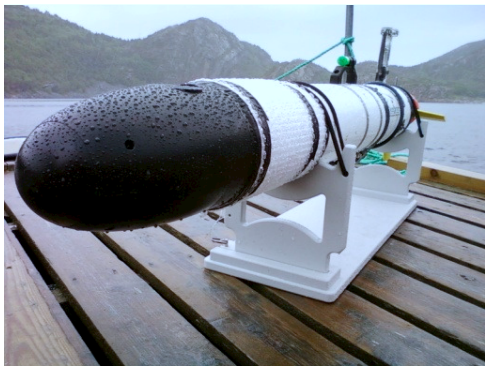

Figure 4: The AUV, programmed by Calpoly CS freshmen, at Avila Bay.

Grade	Fall 2009 %	Winter 2011 %	Difference
A	28.51%	38.99%	10.48%
B	30.12%	30.82%	0.70%
C	16.87%	15.72%	-1.14%
D	9.24%	3.77%	-5.46%
F/W	15.26%	10.69%	-4.57%

Table 1: Performance in CSC 101 (123 majors)

3. ASSESSMENT

We empirically studied the impact of the CSC 123 course by analyzing student grades and enrollments in first-year courses.

3.1 Grades

The first null hypothesis N_1 was that students taking CSC 123 would perform the same in first year computing courses as in the previous year without CSC 123. The independent variable was students taking CSC 123. The dependent variables were final grades in CSC 101 and CSC 102 in the *typical* quarter (i.e. on the normal schedule when students are expected to take the course). The control group consisted of all students who took CSC 101 and CSC 102 in Fall 2009 and Winter 2010 respectively. The experimental group was all students who took CSC 123, CSC 101, and CSC 102 in Fall 2010, Winter 2011, and Spring 2011 respectively. There were no substantial differences in the material covered in CSC 101 and CSC 102 between the two years, other than the introduction of CSC 123 which moved the typical CSC 101 and CSC 102 sequence from Fall/Winter quarters to Winter/Spring.

A total of 249 students took CSC 101 in Fall 2009, and 209 students took CSC 101 in Winter 2011. Of the 209 Winter 2011 students, 159 or 76.1% also took CSC 123 (i.e. they were computer science, software engineering, or computer engineering majors). Table 1 reports the percentage of students earning A's (A and A-), B's (B+, B, B-), C's (C+, C, C-), D's (D+, D, D-), and F/W (failing or withdrew from course) for the offering of CSC 101 which is *typical*, or on schedule, for Computer Science, Software Engineering, and Computer Engineering Majors. This data indicates that the number of students earning A's in CSC 101 increased by over 10%, while the number of students failing CSC 101 decreased by over 4%. These differences were both statistically significant using the chi-squared test at a 5% significance level. The Fall 2009 numbers include all students who took CSC 101 (CSC/SE/CPE majors and non-majors), whereas the Winter 2011 numbers include only the students who took both CSC 123 and CSC 101 (i.e. only CSC/SE/CPE majors). For comparison, Table 2 reports the differences when including all students, majors and non-majors, in both Fall 2009 and Winter 2010. The differences are similar.

Grade	Fall 2009 %	Winter 2011 %	Difference
A	28.51%	38.38%	9.76%
B	30.12%	32.54%	2.42%
C	16.87%	14.35%	-2.51%
D	9.24%	3.83%	-5.41%
F/W	15.26%	11.00%	-4.26%

Table 2: Performance in CSC 101 (all students)

Grade	Winter 2010 %	Spring 2011 %	Difference
A	33.68%	28.36%	-5.33%
B	28.95%	36.57%	7.62%
C	20.00%	24.63%	4.63%
D	10.00%	5.22%	-4.78%
F/W	7.37%	5.22%	-2.14%

Table 3: Performance in CSC 102 (123 students)

Grade	Winter 2010 %	Spring 2011 %	Difference
A	33.68%	29.89%	-3.79%
B	28.95%	34.24%	5.29%
C	20.00%	23.91%	3.91%
D	10.00%	5.98%	-4.02%
F/W	7.37%	5.98%	-1.39%

Table 4: Performance in CSC 102 (all students)

A total of 190 students took CSC 102 in Winter 2010, and 184 students took CSC 102 in Spring 2011. Of the 184 Spring 2011 students, 134 or 72.8% also took CSC 123 (i.e. were computer science, software engineering, or computer engineering majors). Table 3 reports results from CSC 102, comparing Winter 2010 and Spring 2011, the *typical* quarters for CSC/SE/CPE majors. Similar to CSC101, fewer students who took 123 failed CSC102. However, unlike CSC101, fewer students who took 123 received A's. Similar results are seen in Table 4 which reports all students, majors and non-majors in CSC102.

Although students who took CSC 123 performed significantly better in CSC 101 than students who did not take CSC 123, we are unable to reject the null hypothesis and claim that students performed better in first-year courses when they take CSC 123.

3.2 Retention

The second null hypothesis N_2 was that students taking CSC 123 would enroll in the second first year computing course (CSC 102) at the same rate as in the previous year without CSC 123. The independent variable was students taking CSC 123. The dependent variable was students in CSC 102 as a percentage of students in CSC 101 in the previous quarter for the *typical* quarter (i.e. on the normal schedule when students are expected to take the course).

The control and experimental groups were the same as for the previous section.

In the control group, 190 students took CSC 102 in Winter 2010 following the 249 students who took CSC 101 in Fall 2009, for a rate of 76.31%. In the experimental group, 184 students took CSC 102 in Spring 2011 following the 209 students who took CSC 101 in Winter 2011, for a rate of 88.04%. In addition, only five of the 205 students who took CSC 123 have elected to change to non-computing majors. We are still waiting on official numbers for the previous years for comparison. Although time will tell if CSC 123 improves retention to graduation, this almost 12% increase in CSC 102 enrollment rates is encouraging.

4. CONCLUSIONS

Our work is still unfinished; we have yet to see the effect of our experiment on students in the existing "103" class that ends the historical first-year sequence, and must instead rely on our impressions, and the reports of our students.

Fortunately, those impressions and reports are positive ones! We are now gearing up for the second iteration of the class, and we've added another track–visual arts–to our existing ones. We consider our experiment a success, and look forward to hearing from others who adopt our approach.

5. ACKNOWLEDGEMENTS

Our work is supported from a variety of sources. We would like to acknowledge Ignatios Vakalis, our department chair, for his vision and commitment to enable these new courses.

The mobile sections used Android phones and labs that were created with support from a Google Faculty Research Award. In addition, the WebIDE system is supported by the National Science Foundation. This material is based upon work supported by the National Science Foundation under Grant No. 0942488. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

Last, we would like to recognize Patricia Marin and John T. Yun, Associate Professors at UC Santa Barbara, for their continued effort in helping us collect first-year assessment data.

6. REFERENCES

[1] O. Alvarado and Z. Dodds. Women in cs: An evaluation of three promising practices. In *SIGCSE*, 2010.

[2] M. Barg, A. Fekete, T. Greening, O. Hollands, J. Kay, and J. Kingston. Problem-based learning for foundation computer science courses. *Computer Science Education*, 10(2):109–128, 2000.

[3] M. Besterfield-Sacre, C. Atman, and L. Shuman. Characteristics of freshmen engineering students: Models for determining student attrition in engineering. *Journal of Engineering Education*, pages 139–149, 1997.

[4] H. Carver, L. Henderson, and et al. Increased retention of early computer science and software engineering students using pair programming. *20th Conference on Software Engineering Education and Training*, pages 115–122, 2007.

[5] J. Coheen and L.-Y. Chen. Migrating out of computer science. *Computing Research News.*, 15(2), 2003.

[6] Crenshaw, Metcalf, Chambers, and Thakkar. A case study of retention practices at the university of illinois at urbana-champaign. In *Proceedings of the 39th ACM technical symposium on computer science education*, pages 412–416, 2008.

[7] T. Dvornik, D. Janzen, J. Clements, and O. Dekhtyar. Supporting introductory test-driven labs with webide. In *Software Engineering Education and Training (CSEE T), 2011 24th IEEE-CS Conference on*, pages 51 –60, may 2011.

[8] A. Fisher and J. Margolis. Unlocking the clubhouse: The carnigie mellon experience. *SIGSCE Bulletin*, 34(2), 2002.

[9] J. Goldfinch and M. Hughes. Skills, learning styles and success of first-year undergraduates. *Active Learning in Higher Education*, 8(3), 2007.

[10] M. Haungs, J. Clements, and D. Janzen. Improving engineering education through creativity, collaboration, and context in a first year course. In *American Society for Engineering Education (ASEE) Annual Conference*, Pittsburgh, PA, 2008.

[11] D. Janzen and H. Saiedian. Test-driven development: Concepts, taxonomy, and future direction. *IEEE Computer*, 40(2):43–50, 2005.

[12] V. Lotkowski, S. Robbins, and R. Noeth. The role of academic and non-academic factors in improving college retention. ACT Policy Report, www.act.org/research/policy/index.html, 2004.

[13] M. Marszalek, A. Snauffer, S. Good, G. Hein, and A. Monte. Mentors improve the college experience of engineering undergraduates. *Proceedings of the 2005 35th ASEE/IEEE Frontiers in Education Conference*, 2005.

[14] W. Pauk. *How to Study in College*. Houghton Mifflin Company, 2000.

[15] L. Williams, E. Wiebe, K. Yang, M. Ferzli, and C. Miller. In support of pair programming in the introductory computer science course. *Computer Science Education*, 12(3):197–202, 2002.