# Mixed Approaches to CS0:

Exploring Topic and Pedagogy Variance After Six Years of CS0

Zoë J. Wood, John Clements, Zachary Peterson, David Janzen, Hugh Smith, Michael Haungs, Julie Workman, John Bellardo, Bruce DeBruhl
Computer Science - California Polytechnic State University
San Luis, Obispo, CA
zwood,clements,znjp,djanzen,husmith,mhaungs,workmen,bellardo,bdebruhl@calpoly.edu

## ABSTRACT

Since 2010, the Cal Poly Computer Science Department has required computing majors to select from a variety of CS0 courses to start their academic year. The broad objective of the course is to attract and retain undergraduates that have no prior experience in CS by using authentic problems that demonstrate the relevance and highlight the role of computers in solving "real world" problems. The course is offered in a variety of thematic "flavors" that leverage a student's pre-existing interests (e.g. in music or art), but all share the common goals of introducing students to the basics of programming, teamwork, and college-level study. While there is overlap in overall goals, the courses vary drastically in topic matter (e.g. robotics, gaming, music, computational art, mobile apps, security) and in pedagogical approach (e.g. principles of design, project-based student driven learning, and traditional topic-based programming modules). The introduction of this CS0 course has increased students' commitment to their major and success in follow-on classes. We present these successes and show that student GPAs in a follow-on object oriented programming course do not vary significantly for the differing subtopics and teaching pedagogies employed in the various flavors. Our report includes examining two student subgroups (those experienced with programming and those new to programming). Our evaluations suggest that the existence and goal of the course matter more than the specific content, with all subtopics and pedagogical approaches performing well.

## KEYWORDS

Introductory programming, computer science education, CS0, retention

**ACM Reference Format:**
Zoë J. Wood, John Clements, Zachary Peterson, David Janzen, Hugh Smith, Michael Haungs, Julie Workman, John Bellardo, Bruce DeBruhl . 2018. Mixed Approaches to CS0:: Exploring Topic and Pedagogy Variance After Six Years of CS0. In *Proceedings of SIGCSE '18: The 49th ACM Technical Symposium on Computer Science Education (SIGCSE '18).* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3159450.3159592

## 1 INTRODUCTION

Student success and retention in introductory computer science courses is a nationwide issue [2]. To address first-year retention, since 2010, the Cal Poly Computer Science Department has required students in computing majors to select from a menu of CS0 courses to start their academic year. These CS0 courses, entitled "CPE123: Introduction to Computing," have a shared goal of introducing students to the basics of computing, teamwork, and college-level study through the lens of some pre-existing, personal interest. The courses vary in topic matter or 'flavor' (e.g. robotics, gaming, music, computational art, security, mobile apps) and employ a mix of teaching pedagogies (e.g. How to Design Programs (HtDP), all team work, traditional topic-based programming modules).

Each freshman cohort of computer science (CS), software engineering (SE), and computer engineering (CPE) majors (approximately 250 students each fall) must take our CS0 course. Each section is limited to an enrollment size of approximately 30 students, meaning that eight to ten sections of CS0 are taught, with five to six of the possible topic flavors being offered. Students select their preferred topic within enrollment and scheduling restrictions. Subsequent courses in the computing curriculum introduce students to programming more formally, and use a more consistent course pedagogy (material, style, and development environment) across the numerous sections (again 8-10). Specifically, students follow our CS0 course with three traditional programming classes, our 'CS1' Introduction to Programming, taught in Python, our 'CS2' Introduction to Object Oriented Programming, taught in Java, and our 'CS3' Introduction to Data Structures, taught in Java. [1]

This paper reports on the positive impact that the inclusion of a CS0 course may have had on student graduation rates and students' attitudes towards computer science. We also explore whether the variance in topic and pedagogy affects student success in follow-on courses. Specifically, we report on the variance in students' GPAs for each CS0 flavor in subsequent courses and pass rates in these classes. Our evaluations suggest that the existence and goal of the CS0 course matters more than the specific content, with all subtopics and pedagogical approaches performing well. In addition, we examine the success of two student subgroups (those experienced with programming and those new to programming), with respect to success in follow-on courses and find that once again, even for these two subgroups, each of the flavors of CS0 do equally well in preparing the students for subsequent courses. These explorations include data from six years of these course offerings (2011-2016).

[1]As of this writing, the ordering of this sequence is changing.

## 2 CS0 GOALS, TOPICS AND PEDAGOGY

In all the CS0 course sections, students engage with core computer science principles through constructivist, open-ended assignments. At a high level, the course aims to: (1) teach core computer science principles and tools, providing a foundation and context for more traditional, introductory CS coursework, (2) explore those core concepts through the lens of some pre-existing interest, contextualizing, and making relevant, often complex topics, and (3) remove barriers commonly attributed to poor CS engagement, including isolation and exclusion, a lack of social relevance, and limitations on creativity. In all sections, no prior programming experience is assumed.

All instructors hold that the following methodologies/ideologies are essential to the course:

- Context based introduction to computing
- Project based learning
- Learn by doing via lab work
- Group work
- Relating CS topics to real-world problem
- Individual creativity in assignments

Outside of these shared goals, each flavor of the CS0 course has been individually created and is taught by a computing faculty member with interest and expertise in that topic. The topic material of the courses varies widely, along with programming environment and pedagogical approach. Over six years, the courses and instructors have varied slightly, but with core topics and teaching methodologies have been fairly consistent. Table 1 highlights some of the teaching methodologies that are valued for each of the flavors. The individual topics and teaching methodologies used in each section are briefly described here, with variance in language and development environment emphasized for quick comparisons, along with instructors individual perceptions of the most distinguishing/unique feature of their flavor of CS0:

### 2.1 Computational Art

Students in this course explore the relationship of visual aesthetics and computation. Students design and write interactive animated computer programs. By applying computational thinking, mathematics, art and design principles, and principles of animation, students create artistic and expressive computer programs. This course is an introduction to computer graphics and animation in two dimensions, the fun and easy way. *The main development environment is Processing (both Java and javascript, p5.js have been used).*

**Course pedagogy:** This course employs a topic-based introduction to computing with each topic rooted in the relevance of the associated artistic computational assignment. For example, students are introduced to loops when exploring the artistic principle of texture and the art movement of pointillism and impressionism. Group work and sharing is used regularly throughout the class to help build community along with pair programming for labs and a final project that can be completed individually or as a team (with most students choosing to work in a team). [15]. *One of the most distinguishing features of this flavor is the strong emphasis on creativity and the creative process related to art, design and personal expression.*

### 2.2 Games

The course on game design explores computing and software development by creating a full-featured 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 focuses on design, teamwork, and using an iterative development process. *Students learn basic programming skills in order to develop a simple 2D game using an object-oriented language, most frequently Java and the Greenfoot game development environment.*

**Course pedagogy:** The course is rooted in an individual student-driven learning process and project-based learning. Teams are formed early in the class and constructed to provide an educational support system for its members. In the first half of the course, students complete a series of structured programming assignments designed to provide a basic understanding of both Java and the Greenfoot game development environment and, in the second half, further advanced topics (pathfinding, collision detection, animation) are explored based on the requirements of their team's game. Students follow an iterative development process cycling through the following steps: prototyping, playtesting, evaluation and refinement. Both user testing and Agile-based progress updates are key components of the course. The course provides traditional lecture-based learning in teaching game design theory and a flipped-model format for programming instruction. *One of the most distinguishing features of this flavor is the large quarter-long project and 15 page paper requirement.*

### 2.3 Mobile computing

The course focused on mobile computing started with Android development; due to an instructor change, it switched to iOS in 2013. Students explore computing and software development by creating apps for Android or iOS-based mobile devices. Students work in teams to imagine, design, prototype, build, test, and launch apps for the Android or iOS market. Students gain hands-on experience with the languages and tools needed to build Android/iOS apps, and test their apps on the appropriate device. *For Android, students learned using AppInventor in the first four weeks, then Java in the final six weeks. iOS students use either Objective-C or more recently, Swift.*

**Course pedagogy:** This course is a mix of traditional topic-based introduction and team project-based learning. Students complete weekly labs in pairs or individually, and complete projects in pairs or larger groups up to five students. *One of the most distinguishing features of this flavor is the fact that students are able to create mobile apps very quickly in first week, then move to professional tools to build an app of their own choosing.*

### 2.4 Music

In the CS0 course focused on music, students write programs that generate music. Early compositions focus on the elements of western harmony. Later ones allow students to explore timbre and digital sound processing. Students work collaboratively in teams to design, develop, and test their programs and compositions. The course focuses 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. *This course*

**Table 1: Some of the key methods employed in the various 123 sections other then the universal methods mentioned. A star denotes that this methodology is considered: "Predominantly" important, while a checkmark denotes "Somewhat important".**

| Pedagogy | Comp. Art | Games | Mobile | Music | Robotics | Security |
|---|---|---|---|---|---|---|
| community building | ✓ | | | ✓ | | ✓ |
| flipped model | | ✓ | | | ★ | |
| individual creativity | ★ | ★ | ★ | ★ | ✓ | ✓ |
| individual learning | | ★ | ✓ | | ✓ | |
| lecture based learning | ★ | ✓ | | ★ | | ✓ |
| pair programming. | | | ✓ | | | |
| student feedback (polling) | ✓ | ★ | | ✓ | | ★ |
| student reflections (journal, etc.) | ✓ | ★ | | | | ★ |
| team projects | | ★ | ★ | ★ | ★ | ✓ |
| test 1st design (Htdp) | | | | ★ | ★ | |
| topic based intro to CS | ★ | | ✓ | ✓ | ★ | ✓ |

*closely follows the 'How to Design Programs'(HtdP) [6] curriculum and uses Racket as a programming language.*

**Course pedagogy:** The HtdP curriculum is built on several pillars. First, it puts forward a concrete design recipe with six steps to help carry students from problem statement to solution. Second, it focuses on test-first design, and students following the design recipe are required to specify a function's mapping from inputs to outputs with executable test cases. Third, it is algebraic, eschewing state and mutation in order to make functions composable and testable. Fourth, it is data-driven, categorizing problems using the data that they consume, and showing how to use the specifications of each kind of data to construct function templates that guide the students' thinking. Midway through the course students are gathered into teams of three or four, with whom they complete their assignments for the remainder of the quarter including a team final project. Students participate in weekly reflection surveys in order to allow them to consolidate knowledge and identify issues. *One of the most distinguishing features of this flavor is the use of the HdtP design recipe.*

## 2.5 Robotics

This course is focused on embedded systems and robotics and is structured such that students explore computing, hardware and software development by designing and implementing an embedded system such as a robot or an Internet of Things solution. Students work in teams to imagine, design, prototype, build, and test their own embedded system solution. *The robot used in the curriculum uses an Arduino microcontroller and the students implement their control software using C.*

**Course pedagogy:** The course includes lectures taught in a flipped format, required labs and a quarter-long design project. The course lecture and lab focus on implementing a wheeled robot. For the flipped lecture format the students are given videos, reading material and quizzes to work on outside of the normal lecture period. Then lecture period is dedicated to individual and group activities that help students master the material which is then used in lab to implement the hardware and software to control their robot. All students also implement a quarter-long design project (for example an embedded system to help the elderly). [14]. *One of the most distinguishing features of this flavor is that the course is taught in a flipped manner, with the quarter long project based on service to others.*

## 2.6 Security

In the CS0 course focused on security, students learn the fundamentals of computer science through a hands-on exploration of computer security. Specific security topics include: cryptography, authentication, software security, web security, social engineering, digital forensics, and network security. The content is presented in a unique way, using techniques drawn from alternate-reality games (ARGs) [7]. ARGs are designed to encourage players to collaboratively uncover and interpret fragments of a story, distributed across multiple forms of media, using the "real world" as its platform. *Modules are designed to be solved using a variety of approaches, including computer programming (e.g. port scanning for hidden web services), social engineering (e.g. fooling the instructor into entering a flag into a student-designed phishing website), and physical security (e.g. picking a lock, used on a locker containing a cell phone used for two-factor authentication).*

**Course pedagogy:** The course is organized into modules that explore a different security topic, supported by a set of core CS principles. Students are organized into groups of four, where team members work together throughout the quarter, and prior experience in CS is accounted for, and distributed among the teams. Instruction follows a process-oriented guided inquiry learning (POGIL) approach, where students are encouraged to explore a topic, with only occasional guidance from the instructor. In addition to technical exercises, students are regularly prompted to reflect on their individual experiences. These responses are used for identifying technical misconceptions, social problems within the group, degenerative cases in the challenges, and bugs in the infrastructure.*One of the most distinguishing features of this flavor is the use of an AR game and narrative based assignements in the instruction.*

## 3 RELATED WORK

The initial development of the CS0 course at Cal Poly is described in an earlier paper [13], with many of the core driving principles of the course founded on related work [1, 5, 12]. For instance, allowing students to start their computing career via different flavors or topics is not unique to our CS0 course [3, 4, 8–10]. Georgia Tech has done excellent work on long term studies of their menu-based introduction to computing [11], examining attitudes towards each track and their varied success.

This work confirms the positive outcomes associated with starting with a context based CS introductory course and suggests a framework for evaluating whether mixed CS0 courses that employ differing topics or pedagogy correlates with the student success in follow-on course. Specifically, we report that there was no statistically significant difference in GPAs in follow-on classes for students in the differing flavored CS0 courses in our setting.

## 4 EXAMINING STUDENT SUCCESS

### Broad success

After six years of teaching our CS0 course, statistics indicate that the course has contributed to positive outcomes in terms of graduation rates, student attitude, and subsequent student grades. All the CS0 course instructors, and indeed the entire CS faculty, believe the course is an important addition to our curriculum.

In terms of the long term success of the CS0 course, graduation rates have increased for all computing majors from those in 2009 (prior to the introduction of the CS0 course). Table 2 shows the increase in graduations rates (4 year and 5 year) for all computing majors. Note that SE graduation rates are more volatile as they are a smaller population with cohort sizes ranging from 31-54 students, versus 79-159 CS majors and 77-124 CPE majors during the relevant graduation time period (2009-2012).

In addition, to increased graduation rates, surveys showed that students' self-efficacy in, and attitudes towards, their major improved. When the students were surveyed after the follow-on data structures course (CS3), of the cohort that did not take our CS0 course (prior to its requirement), only 39% "strongly agreed" that they were excited about computer science, while of those who did take our CS0 course, 47% "strongly agreed" that they were excited about computer science. Similarly, only 76% of those who did not take CS0, but 83% of those who did reported seeing the relevance of CS to their future career.

We also saw an increase in the number of 'A' grades in the follow-on data structures course from 18% for those who did not take our CS0 course to 24% for those who did have the CS0 course. One result that did not show strong improvement was the fact that students still found computer science "harder," "more tedious," "more antisocial," and "less important to society". This hints at the limitation and dangers of selling computer science to be something it is not.

### Success in follow-on courses

Given the broad success of the CS0 course, we further examined whether the topic and/or pedagogy of the different flavors reflects any variance in student success in subsequent courses. Our unique setting, with each major cohort of approximately 250 students taking different flavors of CS0, but then all later taking more consistent follow-on programming classes, allows us to examine whether the topic and pedagogy make a difference in student success in subsequent traditional computer science courses.

In our exploration of students' experiences in our early courses, we also considered ways to examine student experience based on their prior programming knowledge. To this end, we have divided the entering freshmen into two subgroups, those who have programming experience and those who do not. These two subgroups are distinguished based on whether the student has any type of AP computer science score (regardless of the score's numeric value) with 443 students of the total 1834 having an AP CS score reported.

Over the course of 2011-2017, 1835 students took our CS0 course. Of these, 1577 students (87%) either passed CS1 in their first attempt or were given credit for it. The number of students passing CS2 in their first attempt or getting credit for it was 1476 (80%). Note that the second set is not a subset of the first, because some students that failed CS1 in their first attempt passed CS2 in their first attempt.

See Figure 1 for an overview of the pass rates per flavor in the follow-on programming courses, over six years. These figures show that all flavors of CS0 courses are performing similarly well in terms of preparing students to pass the subsequent programming courses. These figures are divided into two populations, those students took and passed the CS AP exam in high school and those who did not.

### Density of GPAs

Early analysis of our CS0 courses shows that overall student GPAs tend to drop from when the students enter the next more formal programming course. This is not surprising since each CS0 course was intended to be an 'easy and fun' introduction to computing. However, it was noted that there was a variance in the GPA drop experienced by students coming from the various sections. Students transitioning from the music flavor consistently experienced the lowest GPA drop (For example, the average GPA drop for all sections 2011, 2012 and 2013 was -.89%, -.59% and -.91% respectively, while the drop for those in the music section for those same years was -.45%, -.33% and -.27%. This is also reflected in the average retention through our CS2 (OOP) class (i.e. those who received a passing grade), with those in the music section passing CS2 at a rate of 76.4%, while the average pass rate for all sections was 71%. This led to some of the initial work to examine student success in the various CS0 flavors.

In considering whether a specific CS0 section was indeed a better course, we examined the GPA distributions of 1154 students who went on to receive a grade in follow-on programming classes (after two quarters in Object Oriented Programming). We also looked at grade variation for the students after only one quarter (in Introduction to Programming). Note that although we have been teaching the CS0 course since 2010, there have been other changes to the intro series courses (i.e. a switch from C to Python in the Introduction to Programming, etc.), however, we primarily are focused on examining student success in follow on programming courses given starting out in a particular CS0 flavor.

Figure 2 shows the GPA density plot for the different CS0 flavors in the follow-on CS1, for cohorts for the years 2011-2016. All grades are reported for students who did not have an AP CS score (as AP CS students do not take this course). Though there are differences in the GPA densities, ANOVA shows no significant variance (p-value

Table 2: Graduation rates before and after the introduction of CS0

| major | 4 year Graduation rates prior | 2010 cohort | 2011 cohort | 2012 cohort |
|---|---|---|---|---|
| computer science | 17.1% | 27.8% | 29.6% | 34.6% |
| computer engineering | 21.8% | 36.4% | 30.6% | 25.4% |
| software engineering | 22.6% | 19.0% | 29.4% | 33.3% |
| majors | 5 year Graduation rates prior | 2010 cohort | 2011 cohort | 2012 cohort |
| computer science | 51.7% | 62.0% | 66.7% | no data |
| computer engineering | 47.8% | 61.8% | 66.7% | no data |
| software engineering | 66.7% | 51.2% | 71.4% | no data |

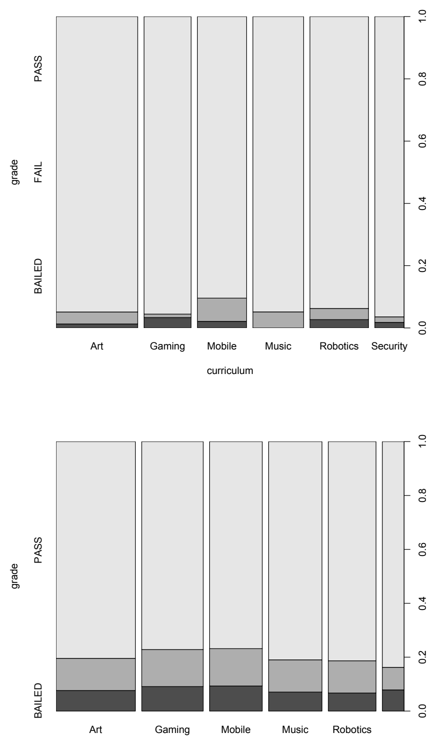

**Figure 1: The top figure shows pass/fail rates for the various CS0 courses for the follow-on two programming courses for students with an AP computer science score. The bottom figure shows the same data but for students with no AP CS score. Dark grey shows the population that did not take a follow on course, mid-grey shows those who failed one of the courses and light grey, the majority, shows those who passed. The width of each bar corresponds to the number of students who took each flavor of CS0, for example, 466 students took the computational art flavor versus 136 students took security (a relatively new offering).**

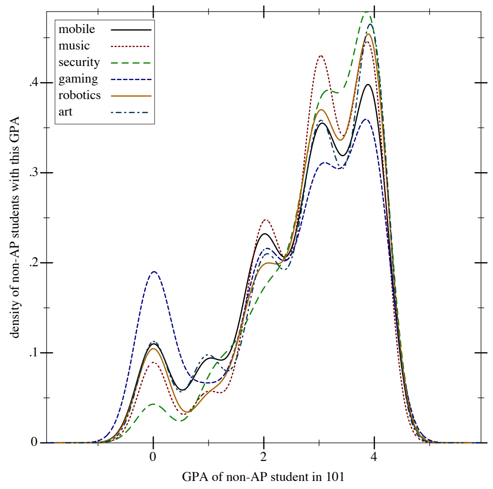

**Figure 2: Density plot of GPAs for students from the various flavored CS0 courses in the follow-on 'Introduction to Programming' course for students with no AP CS score. ANOVA shows no significant variance (p-value of .449).**

of .449), suggesting that students from the different flavored CS0 courses perform similarly in the follow-on programming course. Figures 3 shows the distribution of the grades in our object oriented programming course (two quarters after our CS0 course) for the two groups of students for the the years 2010-2015. Although there is variance between the grade distributions, ANOVA analysis and Chi-squared analysis showed p-values too large to conclude that any particular section's GPA distribution was significantly different then the others. In all of these graphs, students that elected not to take the course are considered to have earned a GPA of 0.0, except for those that went on to take a later course in the first-year sequence, who are assumed to have bypassed this requirement either through an AP score or by taking the course elsewhere.

## 5 CONCLUSIONS AND FUTURE WORK

We have presented our explorations of six years of teaching CS0 as a first course for computing majors. Overall, we have experienced that this course is beneficial to our department, contributing to an increase in graduation rates, improvements in students' attitudes towards their major and overall increase in GPAs in follow-on courses. In addition, the unique setting with faculty able to employ various pedagogy and unique material has created a fertile grounds for faculty growth. In a recent survey of the participating faculty (nine in total), 77% said that they had adopted many to a few different

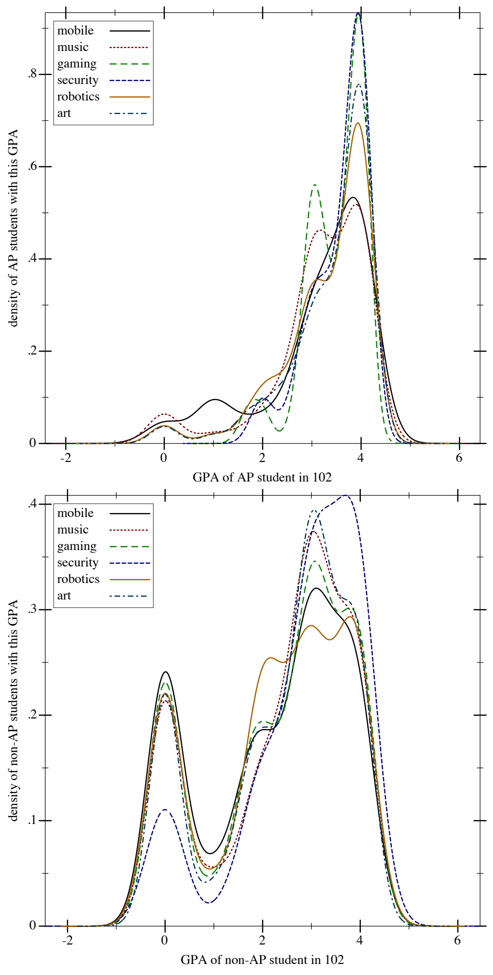

**Figure 3: Density plot of GPAs for students from the various CS0 courses in the follow-on 'Object Oriented Programming' course. The top graph shows, all grades for students who entered college with an AP CS score. ANOVA shows no significant variance (p-value: .815). The lower graph shows all grades for students who did not have an AP CS score. ANOVA shows no significant variance (p-value: .581).**

teaching strategies based on hearing what other instructors were doing in their sections. One of the most commonly listed practices that was adopted was the weekly reflections/journaling. Some of the faculty comments about the structure include: "It is great to exchange ideas and try to adopt practices that work well in other sections without getting stuck debating the technical minutia." and "I like comparing notes and brainstorming pedagogical approaches; I like that we have a potential to measure and evaluate the efficacy of our varied practices". For a field, renown for language wars, it has been liberating for our faculty to do their 'own' thing and see that all of the students can be prepared via very different approaches.

We examined student success in follow-on courses both in terms of pass and fail rates and GPA distribution and have found that no single flavor of our CS0 course is statistically outperforming the others. This finding in many ways is positive as it demonstrates that no single topic or pedagogy for CS0 of the six variations being used is significantly better. This implies that continuing with all the various flavors and teaching methodologies is appropriate and that any school considering CS0 options could select similar topics and pedagogy to the six presented here.

In the future, we would like to have a more refined subgroup selection process as we acknowledge that many students who do not have an AP score, could have substantial computing experience via summer camps, robotics clubs, etc. In addition, we acknowledge the limitation of drawing conclusions from this exploration. Our graduation rates have increased, but so has the overall popularity of computer science. In addition, this examination gave us an opportunity to think about how we can measure student success and whether any of our teaching pedagogies are perhaps more successful than others (certainly, it is appealing to discover if there is one right way to teach intro CS). We wish to continue to expand our understanding of which factors contribute the most to student success and retention.

## REFERENCES

[1] Christine Alvarado and Zachary Dodds. 2010. Women in CS: an evaluation of three promising practices. In *Proceedings of the 41st ACM technical symposium on Computer science education.* ACM, 57–61.
[2] Jens Bennedsen and Michael E. Caspersen. 2007. Failure rates in introductory programming. *SIGCSE Bull. 39* (2007), 32–36.
[3] Elizabeth Bonsignore, Kari Kraus, Amanda Visconti, Derek Hansen, Ann Fraistat, and Allison Druin. 2012. Game design for promoting counterfactual thinking. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems.*
[4] Robert Bryant, Richard Weiss, Genevieve Orr, and Kathie Yerion. 2011. Using the Context of Algorithmic Art to Change Attitudes in Introductory Programming. *J. Comput. Sci. Coll.* 27, 1 (Oct. 2011), 112–119. http://dl.acm.org/citation.cfm?id=2037151.2037177
[5] Zachary Dodds, Ran Libeskind-Hadas, Christine Alvarado, and Geoff Kuenning. 2008. Evaluating a breadth-first cs 1 for scientists. In *ACM SIGCSE Bulletin*, Vol. 40. ACM, 266–270.
[6] Matthias Felleisen, Robert Bruce Findler, Matthew Flatt, and Shriram Krishnamurthi. 2001. *How to Design Programs: An Introduction to Programming and Computing.* MIT Press, Cambridge, MA, USA.
[7] M. Gondree and Z.N.J. Peterson. 2015. This is Not a Game: Early Observations on Using Alternate Reality Games for Teaching Security Concepts to First-Year Undergraduates. In *Proceedings of the Workshop on Cyber Security Experimentation and Test (CSET).*
[8] Ira Greenberg, Deepak Kumar, and Dianna Xu. 2012. Creative Coding and Visual Portfolios for CS1. In *Proceedings of the 43rd ACM Technical Symposium on Computer Science Education (SIGCSE '12).* ACM, New York, NY, USA, 247–252. https://doi.org/10.1145/2157136.2157214
[9] Mark Guzdial. 2006. Teaching Computing for Everyone. *J. Comput. Sci. Coll.* 21, 4 (April 2006), 6–6. http://dl.acm.org/citation.cfm?id=1127389.1127390
[10] Mark Guzdial. 2010. Does Contextualized Computing Education Help? *ACM Inroads* 1, 4 (Dec. 2010), 4–6. https://doi.org/10.1145/1869746.1869747
[11] Mark Guzdial. 2013. Exploring Hypotheses About Media Computation. In *Proceedings of the Ninth Annual International ACM Conference on International Computing Education Research (ICER '13).* ACM, New York, NY, USA, 19–26. https://doi.org/10.1145/2493394.2493397
[12] Susanne Hambrusch, Christoph Hoffmann, John T Korb, Mark Haugan, and Antony L Hosking. 2009. A multidisciplinary approach towards computational thinking for science majors. In *Proceedings of the ACM technical symposium on Computer science education.*
[13] Michael Haungs, Christopher Clark, John Clements, and David Janzen. 2012. Improving first-year success and retention through interest-based CS0 courses. In *Proceedings of the ACM Technical Symposium on Computer Science Education.*
[14] H. Smith. 2016. An Embedded System Design Experience for First Year Computing Majors. In *Proceedings of the 8th annual First Year Engineering Experience.*
[15] Z. J. Wood and J. Workman. 2013. Computational art using Processing for CS0. In *Grace Hopper Celebration of Women in Computing.*