# ADOPTING CS PRINCIPLES IN A BREADTH-FIRST SURVEY COURSE *

*Chris Mayfield*
*Department of Computer Science*
*James Madison University*
*Harrisonburg, VA 22807*
*+1 540-568-3314*
*mayfiecs@jmu.edu*

**ABSTRACT**

With the recent launch of AP CS Principles in 2016-17, many efforts are currently underway to share curriculum resources and prepare new teachers. The community has primarily focused on high school implementations, which have different situational factors than university courses (e.g., amount of class time). In this paper, we present the design of a survey course that aligns with CS Principles and also continues the long tradition of breadth-first introductions to computer science at the college level. We describe the instructional strategies, assessments, and curriculum details, providing a model for how to modify existing CS0 courses. We also outline twelve lab activities that support the computational thinking practices and learning objectives of the AP curriculum framework. The course has run successfully for the past four years at two universities and three high schools via dual enrollment. Initial results suggest that the curriculum has a positive impact on student confidence levels and attitudes toward computer science.

## MOTIVATION

During the late 1980s, the ACM task force on the Core of Computer Science (COCS) developed an "intellectual framework for the discipline of computing and a new basis for computing curricula" [9]. Their purpose foreshadowed that of the CS Principles effort currently underway: to rethink how we can introduce students to computer science in a way that (1) appeals to a much wider audience and (2) doesn't focus solely on

programming. In short, the COCS proposed that the first course in computer science should "contain a rigorous, challenging survey of the whole discipline." However the framework they laid out ultimately called for a three-course sequence of 42 lectures and 35 labs, followed by a complete redesign of the rest of the undergraduate curriculum. This radical vision was met with criticism (see e.g., [17]), and to this day it has not been fully realized.

Nevertheless, many computer science departments have developed and continue to offer a variety of survey courses [5, 11, 12, 13, 15]. These "breadth-first" introductions to computer science are generally designed for non-majors and implemented as CS0, in some cases with an emphasis on computer fluency or general education. A number of well-established textbooks are available for this approach [4, 8, 18]. Their content directly resembles what CS majors learn and practice in an undergraduate degree program [1]. However, in order to broaden participation, we need to "develop new ways to interest potential students that have not fit into prior paradigms of learning CS" [2].

In recent years, the CS Principles community has developed a curriculum framework that outlines six *computational thinking practices* and seven *big ideas* that encompass the study of computer science [6]. These high-level concepts form the basis of 44 specific learning objectives and over 300 *essential knowledge statements*. One major strength of the framework is that it focuses on broadening participation by emphasizing the impact of computing on society and other fields, as well as enabling students to explore and create artifacts of personal interest. Because the curriculum is independent of any programming language, many different versions of the course have been piloted and evaluated [10].

Over 100 institutions previously attested their support of AP CS Principles, but only 70% of them indicated they will be offering a comparable course [3]. Since universities and colleges are not under the same obligation as high schools when it comes to course objectives and AP assessment, it's difficult to say what "comparable" means. Some may adopt the curriculum developed by official pilot universities, and others may create their own variant of the course for special purposes [16]. However a significant number of universities will likely continue to offer their existing CS0 courses independently and miss the opportunity to rally together as a community in support of CS10K [7] and other national efforts.

This paper offers the best of both worlds: we show how to "upgrade" a traditional survey course to align with the CS Principles objectives and assessments. The point of this work is not to be prescriptive, but rather illustrate how CS Principles can be implemented at the university level to serve as an *alternative CS0 or CS1 for majors*. The overall theme of the proposed course is "how to think like a computer scientist" [19]. Students identify relationships between over a dozen sub-fields of the discipline and the seven big ideas of the AP curriculum framework.

## COURSE DESIGN

We have offered this course for the past four academic years, with multiple sections at the university and dual enrollment sections in three high schools. Preliminary results

from course evaluations and institutional research indicate the course is meeting its objectives. The department's main goals for developing this course are:

- Provide our majors with a common language and broad understanding of CS that will help them put the rest of their coursework into a larger context.
- Align with the proposed AP CS Principles course that seeks to broaden participation in computing in K-12 education.
- Give non-majors a unique opportunity to learn how to think like a computer scientist, without having to take a programming-intensive course.

**Instruction**

Survey courses are notorious for being a mile wide and an inch deep. On top of that, the curriculum framework includes over 300 statements of knowledge that is essential for students to acquire during the course (see Figure 1). Though some objectives require significantly more instruction than others, the framework calls for students to learn a great deal of information. (Incidentally, the big idea of "creativity" has the fewest number of essential knowledge statements.) In order to introduce students to such a large amount of concepts, we rely on a flipped classroom [14].

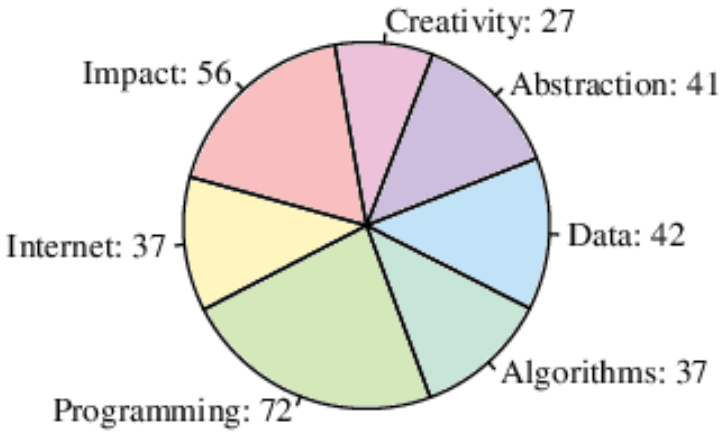


**Figure 1: Number of essential knowledge statements for the seven big ideas of CS Principles.**

Each unit includes 1-2 short videos, some of which we recorded ourselves and others that we gathered from a variety of Internet sources (the latter generally being of higher quality). To encourage the computational thinking practices of analyzing, communicating, and collaborating, we facilitate group activities in class each week that require students to apply what they learned from reading the textbook and watching the videos. Instructors supplement these activities with interactive mini-lectures and demonstrations.

Outside of class, students are required to participate in weekly online discussions related to societal and ethical issues of computing. Each discussion topic is aligned with the current unit of the course. For example, in the chapter about data storage (logic gates

and binary) students discuss a scenario where rounding errors in floating-point arithmetic lead to extensive damage and loss of life. Later in the course when learning about artificial intelligence, students consider to what extent computer algorithms should be used to diagnose patients and offer medical advice in place of a physician.

### Assessment

Because of the large volume of content, the course relies on weekly, 25-minute quizzes with multiple choice, fill in the blank, vocabulary matching, and free response questions. Each quiz is summative in nature and focuses on a specific area of computer science, allowing us to zero in on the essential knowledge statements described in the CS Principles framework. In preparation for the quiz, students complete in-class exercises, online discussion posts, and lab assignments. These activities provide frequent feedback and low-stakes formative assessments.

We also adopted the two performance tasks developed by the College Board as our midterm and final projects. The "Explore" task helps put the first half of the semester into context as students study an innovation of their choice. The "Create" task ties together the second half as students write programs to showcase what they have learned in the course. In order to implement these performance tasks in a university setting, we needed to modify some of the requirements. For both tasks, students work on them primarily outside of class time (since the specified 20 hours would consume nearly half of the semester). We also require interactive demos, rather than have students record and edit videos of their work in advance.

## CURRICULUM

As shown in Figure 2, the course is divided into two main parts. The first half of the semester introduces the hardware and systems side of computer science, beginning with logic gates and culminating with the Internet. The second half focuses on software and data, using Python to illustrate and apply new concepts each week. Students complete the two performance tasks (of the AP assessment) as their midterm and final project. Each of the individual units lasts one week and focuses on a single field of computer science. The online discussions take place throughout the entire course.

| | Creativity | | | | | | | Abstraction | | | | | | | Data | | | | | | | Algorithms | | | | | | | Programming | | | | | | | Internet | | | | Impact | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | LO 1.1.1 | LO 1.2.1 | LO 1.2.2 | LO 1.2.3 | LO 1.2.4 | LO 1.2.5 | LO 1.3.1 | LO 2.1.1 | LO 2.1.2 | LO 2.2.1 | LO 2.2.2 | LO 2.2.3 | LO 2.3.1 | LO 2.3.2 | LO 3.1.1 | LO 3.1.2 | LO 3.1.3 | LO 3.2.1 | LO 3.2.2 | LO 3.3.1 | LO 4.1.1 | LO 4.1.2 | LO 4.2.1 | LO 4.2.2 | LO 4.2.3 | LO 4.2.4 | LO 5.1.1 | LO 5.1.2 | LO 5.1.3 | LO 5.2.1 | LO 5.3.1 | LO 5.4.1 | LO 5.5.1 | LO 6.1.1 | LO 6.2.1 | LO 6.2.2 | LO 6.3.1 | LO 7.1.1 | LO 7.1.2 | LO 7.2.1 | LO 7.3.1 | LO 7.4.1 | LO 7.5.1 | LO 7.5.2 |
| Online Discussions | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | x | x | x | x | x | x | x | x | x | x | x |
| U1: Introduction | | | x | | | | | | | x | x | | x | | | | | | | | x | | | | | x | | x | | | | | | | | | | x | | | | | | |
| U2: Data Storage | | | x | x | | x | | x | x | x | | x | x | | | | | | | x | | | | | | | | | | | | | | | | | | | | | | | | |
| U3: Program Execution | | | | | | x | | x | x | | x | x | x | x | | | | | | | | x | | | | | | | | x | | x | x | | | | | | | | | | | |
| U4: Operating Systems | x | | x | x | | x | | | | | x | x | | | x | x | | | | | | | | | | | | | | x | x | | | | | | x | | | | | | |
| U5: Computer Networking | x | x | | | | x | x | x | | | | x | | x | | | x | x | | | | | | | | | | | | | | | | x | x | x | | x | x | | x | | | |
| U6: Information Security | | | x | | x | | | x | x | | | | | | x | x | | | | x | | | x | x | | | | | | | | | | | | x | x | | | x | x | | | |
| PT1: Explore/Impact | x | x | x | x | | x | | | | | | | | | | | | | | x | | | | | | | | | | | | | | | | | | x | | x | x | x | x | x |
| U7: Algorithms and Python | | | x | | x | x | | | | | x | x | | | | | | | | | x | x | | | | x | | x | x | x | x | x | x | | | | | | | | | | |
| U8: Programming Languages | x | x | | x | x | | | x | | | x | x | x | | | | | | | | x | x | | | | | x | | x | x | x | | | | | | | | | | | | |
| U9: Software Engineering | | | | x | | x | | | | | x | | x | x | | | | | | | | | | | | | | x | | | x | x | | | | | | x | | x | | | | |
| U10: Data Structures | | | | | | x | | x | x | | | | x | x | | | x | | x | | | x | x | | | | | | | | x | x | x | | | | | | | | | | | |
| U11: Database Systems | x | | x | | x | | | | | | | x | | | x | x | | x | x | | | x | | | | x | | | x | | | x | | | | | | x | | x | x | | x | |
| U12: Artificial Intelligence | x | | x | x | x | x | | | | | | x | | x | | | | | | | x | x | x | | x | | | x | | | x | x | | | | | | | | | | | |
| PT2: Create/Program | | x | | x | x | | x | | | | x | x | | | | | | | | | x | x | | | | x | | | x | x | x | x | x | | | | | | | | | | |

**Figure 2: Alignment of course units to CS Principles learning objectives.**

Although our approach is not bound to a specific textbook, we follow Brookshear and Brylow fairly closely in our current implementation [4]. The main contribution of this paper is our adaptation of this kind of material to CS Principles.

In short, we introduce students to the field of computer science in a bottom-up manner, from concrete details about computer hardware to abstract theories of computation. Students work in teams of 3-4 to explore a different sub-field of CS each week. We have developed new lab activities to accompany each of the twelve units. The labs are what makes this course well-suited to teach the computational thinking practices outlined in the curriculum framework. We also adopted (and simplified) the two AP performance tasks for students to collaborate on midterm and final projects of their choice.

All instructional materials including videos, activities, labs, performance tasks, and rubrics are freely available on the course website: https://w3.cs.jmu.edu/cs101/. Quizzes, solutions, and other materials are available to instructors upon request. The textbook also has instructor's guides, PowerPoint slides, and test banks for each chapter.

## REFLECTION

This course was developed at a teaching-oriented, comprehensive public university with about 19,000 undergraduate and 2,000 graduate students. The CS department has 17 full-time faculty and over 500 majors, and each semester we offered 2-3 sections with 25-30 students each. Currently the survey course is an elective, but we are seriously considering making it required for all CS majors. At the time of writing, over 200 students have taken the course, of which about 45% were CS majors and 18% were female.

We have not yet conducted a rigorous, controlled study of the course's impact on student retention and future outcomes. However, we do have a good amount of qualitative data that suggests the course is moving in the right direction. Nearly every student who has taken the course had something positive to say about it. When asked on a survey at the end of the course, “How has taking [CS 101] benefited you?”, student comments included:

“I'm more comfortable working with not only computers, but the software we've used like Wireshark, IDLE, and SQLite.”(CS major, female)

“It gave me a more in-depth knowledge in the field … by teaching material that can actually help you in other CS classes.” (CS major, male)

In terms of recruitment and broadening participation, responses included:

“Taking this class has taught me a ton of valuable information. I took this class to see if I wanted to switch majors to CS, and now I do.” (Nursing major, male)

“[CS 101] has increased my interest in the computer science field. I have enjoyed each chapter and definitely want to continue studying it.” (Undeclared major, female)

“I didn't think I would actually enjoy CS, but this course made me realize that I actually do, especially programming.” (Philosophy major, female)

The survey course has also helped a number of students realize that computer science is not for them-not because they were unable to succeed, but because they had a better understanding of what it entails beyond programming.

"I have learned that CS is not for me, but I find it very fascinating." (Undeclared major, male)

Overall, the course has been well received by students and colleagues. At least one other university (that we know of) has adopted the curriculum for their CS0 course. As part of our ongoing work, we are collecting new data to assess how taking this course impacts learning outcomes and student confidence in subsequent courses. One of the strengths of our curriculum is how it highlights many different levels of abstraction in computer science. Few if any CS Principles courses introduce students to the concepts of machine language and pointers. We argue that such topics are necessary for students to have a correct mental model of the machines underlying not only their programs, but the digital world in which they live. Our hope is that many departments will find this approach to be helpful and desirable for teaching CS Principles at the college level.

## REFERENCES

[1] ACM/IEEE-CS Joint Task Force on Computing Curricula, *Computer Science Curricula 2013*, New York, NY: ACM Press and IEEE Computer Society Press, 2013.

[2] Arpaci-Dusseau, A., Arpaci-Dusseau, A., Astrachan, O., et al., Computer science principles: analysis of a proposed advanced placement course, *Proceedings of the 44th ACM Technical Symposium on Computer Science Education*, 2013.

[3] Astrachan, O., Briggs, A., Cuny, J., Diaz, L., and C. Stephenson, Update on the CS principles project, *Proceedings of the 43rd ACM Technical Symposium on Computer Science Education*, 2012.

[4] Brookshear, J., Brylow, D., *Computer Science: An Overview*, Upper Saddle River, NJ: Pearson Education, 2015.

[5] Close, R., Kopec, D., Aman, J., CS1: perspectives on programming languages and the breadth-first approach, *Journal of Computing Sciences in Colleges*, 15, (5), 228-234, 2000.

[6] College Board, AP CS Principles Curriculum Framework, New York, NY: 2016.

[7] Cuny, J., Transforming computer science education in high schools, *IEEE Computer*, 44, (6), 107-109, 2011.

[8] Dale, N., Lewis, J., *Computer Science Illuminated*, Burlington, MA: Jones and Bartlett, 2016.

[9] Denning, P., Comer, D., Gries, D., Mulder, M., Tucker, A., Turner, A., Young, P., Computing as a discipline, *Communications of the ACM*, 32, (1), 9-23, 1989.

[10] Garcia, D., Astrachan, O., Brown, B., et al., Computer science principles curricula: on-the-ground; adoptable; adaptable; approaches to teaching, *Proceedings of the 46th ACM Technical Symposium on CS Education*, 2015.

[11] Howland, J., Lewis, M., Hicks, T., Pitts, G., A breadth-first companion for the CS I course, *Journal of Computing Sciences in Colleges*, 18, (4), 11-15, 2003.

[12] Huang, T., Briggs, A., A unified approach to introductory computer science: can one size fit all?, *SIGCSE Bulletin*, 41, (3), 253-257, 2009.

[13] Lu, B., Conley, M., Klein, A., Teaching true computer science principles to the general student, *Journal of Computing Sciences in Colleges*, 29, (5), 233-239, 2014.

[14] Maher, M., Latulipe, C., Lipford, H., Rorrer, A., Flipped classroom strategies for CS education, *Proceedings of the 46th ACM Technical Symposium on Computer Science Education*, 2015.

[15] McFall, R., DeJongh, M., Increasing engagement and enrollment in breadth-first introductory courses using authentic computing tasks, *Proceedings of the 42nd ACM Technical Symposium on CS Education*, 2011.

[16] Page, R., Gamboa, R., A more formal approach to "computer science principles", *Proceedings of the 44th ACM Symposium on Computer Science Education*, 2013.

[17] Pratt, T., Upgrading CS1: an alternative to the proposed COCS survey course, *Proceedings of the 21st ACM Symposium on Computer Science Education*, 1990.

[18] Schneider, G., Gersting, J., *Invitation to Computer Science*, Boston, MA: Cengage Learning, 2013.

[19] Wing, J., Computational thinking, *Communications of the ACM*, 49, (3), 33-35, 2006.