SIIP Implementation: Revising the CS Introductory Programming Sequence

Geoffrey Challen (Teaching Associate Professor)
G. Carl Evans (Teaching Assistant Professor)
Margaret Fleck (Research Associate Professor)
Michael Nowak (Teaching Assistant Professor (Starting 6/1/2020))
Michael Woodley (Clinical Assistant Professor)
Craig Zilles (Associate Professor)

Problem Statement

When CS 126 (Software Design Studio) was added to the CS degree program its primary goal was to solve a specific problem that had been observed with the previous sequence of introductory courses. Students who began learning to program for one semester in CS 125 (Introduction to Computer Science) were then left with no opportunities to continue to practice programming during the spring semester (when they took CS 173 (Discrete Structures), a theory course with no programming content) and probably over the following summer. So when they arrived in CS 225 (Data Structures)—an important course and a gateway to our upper-level electives—their programming abilities were rusty. This problem was exacerbated by the fact that CS 225 used and continues to use the programming language C++, a language that students arriving from CS 125 were unfamiliar with. As a result, CS 126 was added to the degree program with three primary goals: keep students programming, introduce them to C++, and provide an opportunity to learn how to design larger pieces of software earlier in the program[1].

Unfortunately, while CS 126 has proven to be an effective bridge to CS 225 for students that take it, not all students are allowed to take it. To help teach design CS 126 makes heavy use of code review done in small groups, and so since its creation enrollment in CS 126 has been limited to CS majors[2]. But only 20% of students that take CS 125 are CS majors. Many are students who are trying to transfer into the department, want to pursue a minor, or are just curious about CS, i.e. non-majors who want to proceed through CS 225 and possibly beyond. For them, CS 126 is not solving the problem it was intended to solve.

[1] Many students would reach the software development course later in the program and either report that they had already had similar experiences in industry, or that they wished that they had seen that design material earlier.
[2] Throughout this document CS majors includes both CS Engineering and all students in the CS+X programs.

This has multiple negative effects—both on the design of our introductory courses and on the students that take them. CS 125 cannot depend on students taking CS 126 afterward, and so must attempt to prepare students to take CS 225 without any further programming practice. CS 225 cannot rely on students taking CS 126 beforehand, and so has continued to need to teach almost a month of C++ programming content, reducing its ability to cover core course material. Non-CS-majors are upset by the fact that CS majors get extra preparation for CS 225, particularly because doing well in CS 225 is important for students transferring into the major. And CS Majors are upset by the fact that they have to take an extra class before CS 225 that non-majors do not need to take.

Proposed Changes

Together our team has proposed curricular changes that will help address the problem described above while also improving the CS introductory programming sequence.

We propose to solve these problems by removing the CS major restriction from CS 126, creating a new course numbered CS 128. This will create a consistent pathway into CS 225 for all students—both majors and non-majors. It will allow CS 225 to rely on a more consistent set of inputs, and move away from teaching programming and towards more theoretical content. It will allow CS 125 to rely on a more consistent downstream pathway, and more closely coordinate with CS 128 to better prepare students for CS 225. And this change will eliminate the perceptions of unfairness among majors and non-majors, since all students will be required to take the same series of CS 225 prerequisites.

Unfortunately, CS 126 as it is currently designed will not immediately scale to the number of students we will need to accommodate to remove the major restriction. Specifically, it is the design components of CS 126 that require high staffing levels and space utilization that will not scale to a larger student population.

To preserve the design and practical software development components of CS 126 our proposal refactors them into a new one credit-hour course: CS 222. CS 222 will be restricted to CS majors, but it will not be required as a prerequisite for CS 225 or any other CS course. CS 222 will require CS 126 as a prerequisite, so that students will reach it a bit later in the program but it will not need to teach programming. Instead, CS 222 will focus on teamwork and software development. Students will do semester-long independent projects in teams under the supervision of course staff mentors.

As a result of these changes, CS 225 can rely on a more consistently prepared set of student inputs. This in turn will enable us to proceed with long-planned improvements to CS 225, with the removal of introductory C++ content from the course allowing a more rapid progression into the core data structures and algorithms content.

Summary

Our proposal affects these courses, which are currently configured as follows:

- CS 125: 4 credit hours, open to all students
- CS 126: 3 credit hours, limited to CS majors, requires CS 125
- (CS 225: 4 credit hours, requires CS 126 for CS majors but not for non-majors)

Existing credit hours will be redistributed as follows, with changes marked in bold:

- CS 125: **3 credit hours**, open to all students
- CS 128: 3 credit hours, **open to all students**, requires CS 125
- **CS 222: 1 credit hour, limited to CS majors, requires CS 128**
- (CS 225: 4 credit hours, **requires CS 128 or ECE 220; and CS 173**)

The proposal will result in changes to all four courses, some minor, some substantial:

- CS 125: reduce workload by 1 credit hour and redistribute some material to CS 128
- CS 128: significantly redesign to the point of representing a new course
- CS 222: borrows the code review components and hopefully a good portion of the staff from CS 126
- CS 225: can accelerate by removing C++ instruction from the beginning of the semester

Timeline

Our plan has been approved by the full CS faculty. The approved plan readies the new sequence for freshman entering in Fall 2021.

While we had originally planned to run a small prototype of CS 128 in Fall 2020, many of our team are involved in Covid-19 preparations this summer. Instead, we will take Fall 2020 to help familiarize incoming team member Michael Nowak with the existing courses and begin work on a CS 128 prototype to launch in Spring 2021.

- Fall 2021
    - CS 125 credit hour requirement dropped to 3 credit hours and now fully compatible with the modifications to CS 128.
    - CS 128 offered in its new version to both majors and non-majors.
    - CS 222 offered to a small number of students—either incoming majors who proficiency through CS 125 and CS 128, or returning students (see below).

    - **All students starting CS 125 in Fall 2021 or later will be required to complete the new sequence of introductory courses.**
- Summer 2021
    - Development on CS 128 continues incorporating feedback from the Spring 2021 prototype offering.
    - Development of CS 222 begins in earnest to prepare the first small section to run in Fall 2021.
- Spring 2021
    - CS 126 continues to be taught as currently designed to majors.
    - CS 128 offered as a prototype under the CS 199 number to non-majors only. Enrollment can be capped as it is not yet required for all students entering CS 225.
- Fall 2020:
    - CS 128 development continues toward the goal of a Spring 2021 prototype.
    - Incoming team member and new CS teaching faculty Michael Nowak leads the work on CS 128 while also becoming familiar with the existing sequence consisting of CS 125, CS 126, and CS 225.
- Summer 2020:
    - Development begins on infrastructure and materials to support CS 128 and CS 222.

Support from SIIP

Support from SIIP will allow us to augment our plan above with support for one specific educational objective: increasing opportunities for students to get direct feedback on their code from human evaluators across the entire introductory programming sequence, including both our large and small courses.

Human feedback on code is important. While autograding systems are an extremely valuable pedagogical tool, they are typically limited to determining whether code is correct, not whether it is good. Students may submit correct code that is inelegant in a variety of ways—potentially inefficient, exposing gaps in their knowledge, or possibly unmaintainable. While there are existing automated code quality tools that we will also explore and deploy across our introductory programming sequence, there will probably always be aspects of code quality that machines cannot properly evaluate, leaving a role and a requirement for direct human evaluation.

While we do not anticipate that we will continue to be able to offer personalized code review in CS 128 which replaces CS 126, we do think that there are opportunities to use technology to facilitate more limited forms of code review at scale across the entire introductory sequence. One possible approach uses crowdsourcing. Students first submit answers to a programming problem that are autograded. Next, their submissions are

distributed to a group of evaluators that includes both students and course staff. Each evaluator provides an evaluation of the code by answering a small number of multiple choice questions—for example, “Does the code shown use appropriate variable names?” Scores submitted by the course staff are used as ground truth for each piece of code. Students are then graded in two ways—both on the scores that their code received from the human evaluators, and second on their ability to rate other students' code accurately based on their match to the scores submitted by the course staff.

Evaluation Plan

We plan to evaluate our new introductory curriculum in two ways. First, how well each of the individual courses works. And second, how well the courses work together to meet learning goals across the entire course sequence.

To evaluate the courses individually we will use standard metrics including ICES scores and student surveys. We will also consider the resources required to teach CS 128 at scale and whether they are commensurate with the other courses in the sequence.

To evaluate the courses as a group, our primary focus will be on the effects that our changes have on student preparation for CS 225 and whether this supports the continued improvement of that course. We anticipate that non-majors[3] proceeding to CS 225 should be more prepared than they were in the past. We can measure this both by examining their outcomes compared to prior cohorts, but also by assessing their preparation level at the beginning of CS 225. We hope to encounter both better familiarity with C++ and also a higher comfort level with command line tools and environments, among both cohorts of majors and non-majors but particularly among non-majors.

However, at the same time a better and more uniformly prepared group of students entering CS 225 should allow the course to proceed with a long-stalled plan to remove C++ instruction from the beginning of the courses and move more quickly in the core data structures and algorithms that represent the conceptual focus.

It is important to note that a more rapid pace at the beginning of the course will tend to cause some students to struggle even if they are, as a group, more prepared due to CS 128. However, we anticipate that the changes to CS 225 will proceed slowly enough for us to measure the impacts of CS 128. In fact, we hope to be able to proceed with the changes to CS 225 exactly because we can determine that students are more prepared on entry and thereby ready to see complex topics earlier.

[3] Exclude ECE students, who have and will continue to have some of the same preparation we are trying to provide to other non-majors

Budget Request

Our budget request for 2020–2021 comprises $16,800[4] to support two undergraduate developers during Summer 2020. The undergraduate developers will help build out the infrastructure and new assignments required to support the changes to CS 128 and the surrounding courses. Supported students will also work on creating a framework to support collaborative code review as described above. Student developers will be primarily supervised by new CS teaching faculty member Michael Nowak, who we anticipate will be responsible for offering CS 128 both as a prototype in Spring 2021 and the first time it is offered as an official part of our degree program the following semester.

We also expect to receive in-kind support from the CS department, definitely including continuing support for these student developers during the 2020–2021 academic year and potentially also including summer salary support for Michael Nowak.

Our Team: Biosketches

Our team brings together multiple CS instructors with experience teaching all of the courses directly involved in the revision (CS 125, CS 126, CS 225) and CS 173, which is taken alongside CS 126 and coordinates closely with both CS 126 and CS 225.

Geoffrey Challen

Geoffrey Challen is a Teaching Associate Professor in Computer Science. He has taught CS 125 at the University of Illinois since Fall 2017.

G Carl Evans

G Carl Evans has been a Teaching Assistant Professor in Computer Science since Fall 2017. Immediately prior to that he completed both his undergraduate and graduate work with the University of Illinois CS department. As a professor he has taught many of the courses involved in this revision, including CS 126, CS 173, and CS 225.

Margaret Fleck

Margaret Fleck is a Research Associate Professor in Computer Science. She has taught Theory of Computation (CS 273) and also Artificial Intelligence (CS 440). For the last decade, she has been course coordinator for CS 173: Discrete Structures, which is taken in parallel with CS 128, the central course in this revision.

---

[4] Calculated as 2 students * 15 weeks / summer * 40 hours / week * $14 / hour = $16,800

Michael Nowak

Michael Nowak joins the CS department as a Teaching Assistant Professor in June 2020. Formerly, he has taught seven semesters of *Introduction to Program Design and Concepts (CSCE-121)* in the Department of Computer Science and Engineering at Texas A&M University (TAMU). TAMU's 121 is taught in the C++ programming language to 100-students per section and is a course that was originally designed by the creator of the C++ language (Bjarne Stroustrup). Recently, Michael has participated in a collaborative teaching initiative towards the revision of this course. The resultant TAMU course, CSCE-120, has content overlap with UIUC's CS 125 and proposed CS 128 course as well as similar pedagogical considerations including auto-grading and code design reviews.

Michael Woodley

Michael Woodley is a Clinical Assistant Professor in Computer Science. He has taught CS 126 or a later-equivalent, CS 242, since 2005.

Craig Zilles

Craig Zilles is an Associate Professor in Computer Science. He has taught CS 125, CS 126, and CS 225, and was one of the original designers of CS 126.

Syllabi for Affected Courses

The post-change syllabi for all affected courses follow for reference.

CS 125: Introduction to Computer Science I

Basic concepts in computing and fundamental techniques for solving computational problems. Intended as a first course for computer science majors and others with a deep interest in computing.
— 3 credit hours. Required. Prerequisites: Math 112 or an adequate ALEKS placement score as described at http://math.illinois.edu/ALEKS/, demonstrating knowledge of the topics covered in Math 112.

Learning Objectives

1. Formulate useful algorithms that solve real problems and can be implemented and run on a computer.
2. Read and understand simple computer programs—determining intention, tracing runtime behavior, identifying bugs and errors that might occur, and estimating computational cost and complexity.
3. Create computer programs that solve a variety of problems using appropriate techniques, including imperative and recursive approaches, and utilizing simple data structures such as arrays, lists, maps, and records.
4. Write computer programs that represent and manipulate different types of information—such as text, audio, and visual data.
5. Develop and debug programs using industry-standard tools and best practices, including visual editing and debugging and source version control.
6. Understanding and implementing classic computer science algorithms such as sorting and searching data while understanding their behavior and running time.

Assessments and Environment

- Daily small problems in Java/Kotlin completed in the browser
- Weekly CBTF quizzes including both programming and conceptual questions
- Semester-long project completing an Android app in Java/Kotlin using Android Studio

Course Description

CS 125 is an introduction to the concepts and craft of computer science. It teaches students to both think and act like computer scientists. It changes how they approach problems and provide them with powerful tools that they can use to change the world. CS 125 assumes no prior programming experience, but is best suited for students with a strong interest in the subject and who intend to continue on to CS 126 and further courses in computer science.

Topic Schedule

Listed by week.

1. Types, Variables, Data
2. Statements and Expressions
3. Functions
4. Branching and Looping
5. Errors and Exceptions
6. Strings and Text
7. Records
8. Arrays
9. Lists
10. Maps
11. Algorithm Analysis
12. Recursion
13. Sorting
14. Searching
15. Functional-Style Programming

CS 128: Introduction to Computer Science II

Continuation of CS 125: Introduction to Computer Science I. More advanced concepts in computing and techniques and approaches for solving computational problems.
— 3 credit hours. Required. Prerequisites: CS 125 or ECE 220. Math 112 or an adequate ALEKS placement score as described at http://math.illinois.edu/ALEKS/, demonstrating knowledge of the topics covered in Math 112.

Learning Objectives

Progressions from CS 125 are italicized. Entirely new objectives are in bold.

1. Create computer programs that solve a variety of problems using appropriate techniques, including imperative, recursive, *object-oriented,* **generic, functional, and concurrent approaches**, and using *more sophisticated data structures such as trees and graphs*.
2. Read and understand *medium-size* computer programs—determining intention, tracing runtime behavior, identifying bugs and errors that might occur, estimating computational cost and complexity, *and evaluating basic design decisions*.
3. Develop and debug programs using industry-standard tools and best practices, *utilizing command-line development environments* and source version control.
4. *Understand the features of computers that make them useful for solving problems, including computation, memory, storage, and networking.*
5. *Use object-oriented design to appropriately structure data, model real-world entities, couple state and behavior, extend existing classes, exploit polymorphism, and implement and create useful interfaces and abstractions.*
6. **Parse and generate serialized data to and from standard formats; and manage information from sources including files, databases, and network APIs.**

Assessments and Environment

- Daily small problems in C++ completed at the command line, although a simple visual editor like VS Code may be encouraged
- Weekly CBTF quizzes including both programming and conceptual questions
- Semester-long project completing a web backend in C++ using command line tools and a simple visual editor

Course Description

CS 128 continues where CS 125 leaves off, but assumes an audience of students with a deeper interest in computer science. As a result, the course dives deeper into *how and why* questions rather than just showing students what is possible. Once they complete CS 128 students have a foundation for a deep understanding of how computers actually work and effective software development practices. They know how to use computing tools and begin to understand how some of them work. They know how to use libraries, evaluate their effectiveness, and create their own software with well-defined interfaces. They appreciate the power of computational thinking and its conceptual foundations. Students completing this course successfully will be ready for a software development internship.

Topic Schedule

Listed by week.

1. Introduction to C++
2. Pointers, References, Smart Pointers
3. Compilation and Execution
4. Memory
5. Files and the Filesystem
6. Networking
7. Databases
8. Classes
9. Inheritance and Polymorphism
10. Interfaces
11. Generic Programming Using Templates
12. Functional Programming
13. Trees
14. Graphs
15. Concurrency and Concurrent Algorithms

CS 222: Software Design Studio

Design and implementation of novel software solutions. Problem identification and definition; idea generation and evaluation; and software implementation, testing, and deployment. Emphasizes software development best practices—including framework selection, code review, documentation, appropriate library usage, project management, continuous integration and testing, and teamwork.
— 1 credit hour. Required. Restricted to CS Majors. Prerequisites: CS 128.

Learning Objectives

1. Identify an interesting problem, understand any previous approaches, formulate an innovative solution, and identify its strengths, weaknesses, and other potential threats to its long-term success.
2. Produce and maintain clear and useful internal and external project documentation—including but not limited to website landing pages, high-level online documentation, code-level commenting, design specifications, bug reports, feature requests, and presentations for technical and non-technical audiences.
3. Create a project plan that establishes a timeline and effectively divides tasks between team members based on their interests and abilities. Establish milestones and monitor progress toward them. Adjust project scope, goals, and milestones as needed to account for unforeseen challenges or delays. Maintain effective lines of communication and collaboration between team members.
4. Implement the project using appropriate software engineering tools and best practices: Work from a specification. Write tests first. Design code so that it can be tested. Test code continuously. Use standard formatting and idioms across the entire project. Choose a language and framework appropriate to the problem. Review code as a team. Use version control tools to support these best practices.

Assessments and Environment

- One semester-long project completed in a team of at least three students under weekly supervision by a course staff mentor.
- Students are expected to choose and use languages and tools appropriate to the problem they have chosen to solve.
- Grading is weighted towards how well students engage in the *process*, *not* what each group ends up producing.

Course Description

CS 222 prepares students to work effectively in a modern software development environment. Students join a team, identify a problem, design a solution, and then implement their design. Along the way they follow software development best practices related to code quality, review, and testing; timeline planning; teamwork and team communication; and maintenance of both internal and external documentation. Each group works under the guidance of a mentor—an experienced student able to provide guidance on these topics.

Class Calendar

Note that we anticipate needing only one beginning-of-the-semester class where all students are present to explain how the course works and answer any questions. After that point all meetings will only include one or several project groups and their mentors.

1. Course Expectations and Team Formation
    a. Team formation based on self-reported skill level, language preferences, and other constraints
    b. Mentors choose times based on the same information used for team formation
2. First Mentor Meetings and Initial Project Discussions
    a. Teams should leave this meeting prepared to present a draft project plan one week later
    b. Initial exploratory development work should begin at this point
3. Draft Project Plan
    a. Presented to mentor and one or two other teams for feedback
4. Final Project Plan
    a. Presented to mentor and several other course staff for assessment
5. Project Pitches
    a. Presented to several other teams for assessment and feedback
    b. While the project plan is an inward-facing document, the pitch is for external consumption
6. Environment Review
    a. Mentors assess how well each group has set up a development environment that will support their project goals
7. Framework Review
    a. Mentors assess each group's choice of frameworks and external libraries and how well they support the project
8. Design Review
    a. Mentors assess each group's design and discuss their plans for transforming it into working code

9. Code Review #1
    a. Mentors examine each group's codebase and discuss testing tools and test coverage
10. Timeline and Teamwork Review
    a. Mentors discuss progress toward project milestones and any adjustments that need to be made to ensure a successful finish
    b. Mentors also discuss how each team is working together, including how well they are using source code version control to collaborate
11. Code Review #2 (Peer to Peer)
    a. Mentors and one other group examine each group's codebase and discuss structural and organizational choices
12. Documentation Review
    a. Mentors examine each group's internal and external documentation and establish expectations for the final project assessment
13. Code Review #3
    a. Mentors discuss high-level design decisions with each group
    b. Mentors also review testing coverage and discuss any gaps with the team
14. Peer-to-Peer Review
    a. Group's present and demo their projects to one or two other group's for feedback before the final project presentations
15. Final Project Presentations and Fair
    a. Each group presents their final project to a group of course staff. All aspects of the project are under review, including design, planning, execution, documentation, code quality, test coverage, and teamwork.
    b. Final project presentations are open to the public
    c. Students are also expected to present their work at a final project fair