WIP: Updating CS1 to a 21st-Century Model of Computing

David P. Bunde*, April R. Crockett†, Gerald C. Gannod†, Jaime Spacco*,
Neena Thota‡, and Charles C. Weems‡
*Knox College, USA, {dbunde,jspacco}@knox.edu, 0000-0001-6334-356X, 0000-0001-6955-0754
†Tennessee Tech, USA, {acrockett,jgannod}@tntech.edu, 0000-0003-1050-1557, 0000-0003-1008-7931
‡University of Massachusetts Amherst, USA, {nthota,weems}@cs.umass.edu, 0000-0002-3795-6060, 0000-0002-8535-0258

***Abstract*—This work in progress innovative practice paper documents ways in which current introductory computing courses are designed for an earlier generation of computers. We describe our plans for updating these courses for modern systems and programming practices and share details of the development of exemplar courses that will be adoptable by diverse institutions and programs teaching introductory programming courses.**

I. Introduction

Twenty-first century computation pervasively employs parallel and distributed computing (PDC), making extensive use of networks to access data as well as utilizing multicore processors locally. Users interact with computers through graphical user interfaces and applications are built on frameworks and application programming interfaces (APIs). The development process is test-driven, with a significant focus on security. By contrast, introductory Computer Science (CS) education has focused on single-threaded, sequential programming that provides little connection to these current practices.

In this paper, we document the disconnect between the modern paradigm of computing and current curricula at two levels. First, we present data gathered by conducting focus group discussions with employers. In general, they reported that recent graduates struggle to write modern software, which builds on an established code base, composes solutions from disparate libraries, accommodates remote data repositories, and takes advantage of concurrency.

Second, we survey current textbooks for introductory programming (CS1), finding that they present a sequential model of computing where input-output (I/O) is achieved through the console and local file system. Furthermore, most programs are written from an empty code file and make little use of APIs beyond basic built-in libraries. Clearly, these books have their roots in texts written for previous generations of computers and programmers.

These issues go beyond a single course, but updating the introductory course to better reflect modern systems and practices is an essential first step. As such, we present our initial efforts to develop materials to modernize CS1 to maximize their adoption potential, with an eye towards using multiple effective evaluation methodologies. The remainder of this paper is organized as follows. Section II discusses previous efforts to revise CS1. The next two sections document the need for changes by summarizing our stakeholder survey on the current state of CS practice and curricula (Section III) and our survey of popular introductory textbooks (Section IV). Then we present our efforts to modernize CS1 in Section V. Finally, in Section VI, we draw conclusions and suggest future investigations.

II. Related Work

Many people and groups have previously advocated changes to CS1. An early effort to introduce PDC topics and include more graphics centered on an early introduction to events [4], which led to a CS1 course [5] supported by a textbook [3] that has unfortunately not been updated. At least one other book [8] attempted to show how CS1 could introduce event-based user interfaces, but was also not updated, due to a lack of market interest. Also specifically focusing on early courses is an effort to generate Unplugged activities for PDC topics [17].

Another big push for updating CS1 was Media Computation [12], [13], with the use of images, sound, and video as motivating examples for programming concepts. This overlaps with our idea of modernizing CS1, which also includes graphics and which we also think will motivate students by making the course closer to their experience with computers. Our goals go beyond motivation, however, because modernizing the course will better prepare students for their future careers.

An older effort to change the content was the Objects First movement, which sought to introduce object-oriented (OO) features earlier in the curriculum. The debate between presenting objects earlier or later in CS1 is summarized by Ehlert and Schulte [9], whose study suggests that each approach leads to similar student learning but different student perceptions of topic difficulty. Both approaches are used in the textbooks examined in this paper, including two books by the same author whose main difference is in when OO features are presented [10], [11].

Curricular recommendations have also moved in the direction we suggest. Incorporating PDC topics was the focus of a large curricular recommendation for undergraduates, which included CS1 and other core courses [7]. Many of these recommendations were adopted in CC2001 [14], the joint recommendations by two of the largest CS professional organizations. The newest set of recommendations [6] includes

other features of modern development, including the use of software APIs and removes specific references to console I/O.

III. Stakeholder Survey

In preparation for our curricular development project, we conducted a two-year study that involved four community input meetings with members of twelve stakeholder groups (faculty, administrators, students, industry, government, funding agencies, authors, publishers, professional societies, accreditation bodies, PDC and CS education projects, international education agencies), with 97 total participants. The meetings used a the World Cafe model [2], in which a large-group discussion is split into smaller discussions by different subgroups to seek both different ideas and common themes. The discussions produced hundreds of pages of notes, which were thematically analyzed. The goal was to identify areas where curricula are lacking, what is impeding progress, what kinds of changes and activities are needed to modernize the curriculum, what actors have a role to play in such an effort, and how to make this change.

Employer stakeholders reported that it takes between six months and two years of additional training before new computer science graduates become productive software developers. They expressed frustration that students are still taught a model of computational thinking that is purely sequential and uses only local data, only learn to build everything up from primitives, typically work individually rather than in a team, learn poor interface design and testing methodology, ignore security, and write software that doesn't scale.

Educators reported that there are no course exemplars, curricula, or textbooks for teaching a modern model, and that most instructors lack the expertise to make changes. They said the curriculum is already full, but also observed that it covers many obsolete coding patterns. Students said that they lack the knowledge to demand change. Administrators said change is hard, especially if students and faculty are not asking for it, and it isn't required for accreditation. Textbook authors and publishers reported no market for books using a modern computing model because nobody is teaching such courses. Accreditation bodies and professional societies said they look to faculty for direction. CS education projects observed that there is not enough funding to develop materials and training for a complete revision of the introductory courses, with proper evaluation of the effectiveness of the changes.

IV. Textbook Survey

Now we document how the disconnect between modern computing and current curricula begins in CS1 by surveying current textbooks for the course. From our stakeholder survey, we know that the three most popular introductory programming languages are Java, Python, and C++. For our textbook survey, we focus on Java because it was developed for the internet age, and is the only one of the three that has built-in support for graphical user interfaces (GUI), threads, and networking. Thus, if any introduction to programming textbook were to take an approach that incorporates all of these modern concepts, it would most likely be one written in Java. We selected the books by surveying publisher booths at the ACM SIGCSE Technical Symposium (SIGCSE TS) in 2023 and 2024 to see what was featured. Because this is the largest computer science education conference in the US, all of the major publishers attend. While publishers do not state which of their books is the most widely used, those are the ones they usually display, along with their newest releases. Thus, we believe that the books selected for our survey on this basis represent what the majority of students in the US are currently using to learn to program in Java. This process identified 6 books, listed in Table I.

TABLE I
Java CS1 textbooks being promoted at SIGCSE TS 2024

Java illuminated by Anderson and Franceschi (6th) [1]
Starting out with Java: Control structures through objects by Gaddis (8th) [11]
Starting out with Java: Early objects by Gaddis (6th) [10]
Java software solutions: Foundations of program design by Lewis and Loftus (10th) [15]
Introduction to Java: Programming and data structures by Liang (12th) [16]
Programming fundamentals using Java by McAllister and Fritz (2nd) [18]

Examination of these books shows that students are indeed focused on a sequential, non-networked model of computing. Only two of the books had any discussion of threads and parallelism, one in the very last chapter [18] and one [16] in an online-only supplement. Because of this placement, students are unlikely to see this material and definitely will not spend much time practicing it even if they use the books where it appears. Networking receives similarly light and optional coverage; two books [1], [18] never use the network in a program and one [15] uses it only to load images via URLs. The others include significant networking coverage, but only in high-numbered chapters: two of the books [11], [10] cover accessing a remote database, including the idea of transactions to maintain consistency, and the last [16] also introduces remote method invocation (RMI), servlets, and web services. Notably, the last is a two-course book and all of this material is in its online-only supplement.

The coverage of graphics also varies widely between books, but it is generally less than what students likely expect from modern applications. Three of the books [11], [10], [16] teach graphics only in late chapters. One [1] introduces JavaFX for drawing graphics fairly early (Chapter 4 of 17), but only uses them as an output medium, not introducing GUIs and interactivity until the last two chapters, which are online-only. Another book [15] uses and covers graphics in sections of nearly every chapter, but these are organized as an optional "Graphics Track" that can be completely skipped and thus graphics are not used in the bulk of the book's code samples. The heaviest use of graphics is a game-oriented text [18], which gives an early introduction to graphics; it was the only one to give a graphical example before a console-based one using `Scanner`. It uses a custom graphics library for most

of the book and does include a version of event handling for input, but many of the graphical examples are output-only.

In general, the books focus on small programs, the kind a student can write in their entirety from scratch. They use large libraries for graphics or advanced concepts in late chapters, but otherwise restrict themselves to basic utility APIs (Scanner, Math, File). The Collections framework is used in late chapters if at all. One book [1] does include a small number of "Team projects", which would require more work. It also includes numerous questions that ask students to modify the text's sample programs, giving a taste of working with an existing code base, though these programs are all small.

In terms of development process, only one book [16] teaches a test framework (JUnit) and that's in a late chapter. (Earlier, it writes programs that print the correct answer and the result of a function call, essentially the same idea, but with the user providing the comparison.) Another [1] includes a category of questions on interpreting error messages. Several of them [11], [10], [15] give paragraph-sized pieces of buggy code and ask the reader to identify the error.

V. Approach

As a result of our surveys of stakeholders and CS1 textbooks, we have obtained funding from the US National Science Foundation to develop exemplar courses for the first year of the CS curriculum that incorporate PDC concepts in the model of computational thinking used by students. Our focus is on addressing the large gap between current computing education (with a sequential approach) and the needs of modern applications and workforce (that deal with concurrency, asynchrony, and distributed objects). A robust evaluation framework will gather data on the effectiveness of our implementations.

We have begun work developing a pair of example CS1 courses that illustrate the potential and benefits of modernizing the CS1 curriculum to a 21st century computing paradigm that can be broadly adoptable in a diverse set of contexts. Our exemplar development and evaluation approach is a combination of two development teams and six test teams. The development teams are at two different types of institutions using two different programming languages. One development team is at Knox College, a private, four-year liberal arts college with approximately 1,100 students, where the introductory computer science courses are taught with Java. The other development team is at Tennessee Tech, a public, Ph.D. granting, research university, with approximately 10,000 students, where the introductory computer science courses are taught with C++.

To ensure that the exemplars we develop are suitable for widespread adoption, they will be evaluated by testing teams at a variety of institutions: four public universities, ranging in size from 4,000 to 19,000 students, one community college, and one private college with 1,900 students. Specifically, each development team's materials will be adopted by test teams with a very different profile. The C++ materials developed by Tennessee Tech (a university) will be tested at the private college and community college. The Java materials developed by Knox (a liberal arts college) will be tested at 3 larger public universities. In addition, one of the public universities uses C#, enabling evaluation of conversion to another language. (Note that we chose not to include Python in our initial efforts, because of how its global interpreter lock hinders easy use of concurrency.) We will be training test team instructors on how to incorporate PDC concepts into their CS1 courses using the framework developed by the development teams.

TABLE II
CS1 Content Mapping Between Institutions

Knox Content in CS1	Tennessee Tech Content in CS1
Welcome	introduction to the course & syllabus
variables	Getting computer set up
identifiers	creating, compiling & running a program
operators	introduction & history of C++
data types	purpose & importance of programming
	basic computer hardware & the power wall
conditionals	problem solving, computational thinking, and algorithm design
while loops	procedural design
	types of instructions in a program (input, processing, output)
	parts of a first program
for loops	basic output, formatting output with setw
	basic input
reference types	comments and whitespace
classes	errors and warnings
	debugging and testing programs
classes	Lab 1: getting computer set up, run & compile first C++ program
which loop to use	variables & data types (integer, double, initialization, assignment, const, char, string)
class String	identifier naming
outputs and inputs	arithmetic operators & expressions
	math library functions
	creating random numbers
	string and character functions
static methods / functions	Relational operators & expressions
	if statement
	if else statement
functions practice	multi-branch if else if statement
	switch statement
photoshop jr library using pixel and picture classes	looping concept
	while loop
	user input validation using loops
arrays	nested loops
	incremental programming
	break & continue
	common algorithms using loops
arrays	Concept of using functions
	defining and calling functions

In Fall 2023, one of the first activities of the two development teams was to compare and contrast our CS1 courses. Semester terms at Knox run for 10 weeks, whereas Tennessee Tech uses 15-week semesters. Table II shows the mapping between the two courses, with each color marking shared content. As an example of the differences we identified, it can be seen that Knox introduces classes and objects in week 2, whereas Tennessee Tech spends time at the beginning of the course on basics such as computational thinking and types of instructions and doesn't teach classes until week 14. As another example, Tennessee Tech's CS1 course includes C++ pointers and structures, which don't have direct equivalents in Java. Below we list commonalities between the courses at both institutions, although they may differ in where they are addressed in the courses.

- variables, identifiers, and data types
- strings
- conditionals / branching
- loops and algorithms using loops
- functions

- outputs and inputs
- arrays
- objects

Another Fall 2023 activity was designing evaluation instruments to measure student understanding of PDC concepts. We used these to gather baseline data for the current courses and will use them again on the new versions. The development teams are currently creating exemplars that introduce PDC components into their own CS1 courses, but coordinating so the same concepts are covered. Both courses will remain focused on teaching the foundations of programming and developing basic programming skills such as program understanding, documenting, specifications, analysis, and testing.

The PDC concepts we are introducing in CS1 are in three categories: data parallelism, distributed computing, and event handling. These categories were chosen because we believe that these aspects of modern computing can be introduced even in a low-level course; we want to teach CS1 students a more accurate model of computing, but must also be mindful of their limited background and avoid excessive conceptual or syntactic complexity.

We have also developed preliminary learning goals for each of these areas. For data parallelism, the learning goals are the following:

- Students will demonstrate the ability to identify algorithmic solutions that are representative of data parallelism.
- Students will have the ability to implement algorithmic solutions that are representative of data parallelism.

Note that these can be met even if the implementation does not use parallel processing; our goal is to promote a way of thinking rather than to speed up the code. One way we foresee doing this is to teach for-each or range-based for loops to process a collection of data, which emphasize that each element is processed rather than giving operations to do so sequentially. Students will also use lambda functions with a collection API, again emphasizing the operation rather than the order, and this time forcing the operation on each element to be independent.

For distributed computing, we have the following learning goals:

- Students will demonstrate the ability to identify when a problem solution requires interaction with a remote service.
- Students will be able to develop and implement algorithmic solutions for interacting with a remote service.

For these goals, students will learn to pull data objects from a remote repository, for example using JSON or similar methods, and develop an algorithmic solution for processing and displaying the data.

For event handling, we have the following learning goal:

- Students will have be able to recognize when an algorithmic solution can benefit from the use of multiple threads that communicate via events or similar means of interaction, and implement that interaction.

To meet this goal, students could build a simple GUI that responds to button presses via an event handler, a GUI that uses background processing while remaining responsive to the user, or an application that invokes remote processing and responds to events associated with it returning data or completing.

In Summer 2024, the development teams trained the test teams in how to use our instrumentation to gather data from runs of their existing courses, and also previewed the CS1 changes for early feedback.

In Fall 2024, the development teams will incorporate the new material in their CS1 courses as a trial run and work with the test teams to transfer the exemplars to their six institutions. The development teams will also be working on creating exemplars for the second programming course (CS2) so that it builds on the concepts introduced in the first.

We are using extensive evaluation processes to document the effectiveness of the interventions. The primary evaluation will be of the exemplars' effectiveness in terms of student learning outcomes. Direct approaches use quantitative performance assessments such as: (i) pretest-posttest comparisons of learning; (ii) standardized tests of disciplinary knowledge; (iii) course evaluations (during the semester and at the end of the semester); (iv) surveys of student attitudes (about new pedagogy, curriculum); and (v) analysis of assignments to test conceptual understanding (e.g., concept maps, exams). Indirect approaches use qualitative or mixed method performance assessments such as: (i) focus groups, (ii) structured and open-ended interviews, (iii) surveys that ask students for reflections on their learning, and (iv) class observations and teacher journals/portfolio.

In early 2024, we collected baseline data in CS1. In Fall 2024, we will collect data in CS1 for analyzing the effectiveness of the new material in teaching PDC concepts. We will also collect baseline data in CS2. The baseline surveys ask students to rate their overall experience with computers and programming. It then asks about their experience with varying programming languages and programming constructs such as data types, arithmetic expressions, branching, loops, functions, arrays, and classes/objects. Pointers and structures are also included for C++. The students are then asked to rate their experience with parallel processing, distributed computing, and event handling.

VI. Discussion

In this paper we have presented results of our stakeholder study, which showed that employers are frustrated by new CS graduates' lack of understanding of modern computing models and practices. The study also showed that academic institutions and associated organizations are held back from modernization by multiple factors, but especially a lack of course exemplars. We corroborated these findings by showing that, with few exceptions, popular textbooks for the first programming course continue to present an obsolete model of computational thinking and devote significant space to teaching older patterns.

We then described our effort to develop exemplar modernized CS1 courses that can be broadly adopted. To our knowledge, this is the only effort where such development is being coordinated across two significantly different (size, course length, language, public vs. private) types of institutions, with subsequent testing of the exemplars by six additional teams at an even more diverse set of institutions, all while employing a robust evaluation strategy.

One change in modern computing practice that we are not addressing here is the use of generative AI tools. One of our development teams is attempting to teach CS1 using these tools, following the method used in a recent Python textbook [19], but this is still preliminary and it is not clear whether other teams in our project will adopt this aspect of the course due to questions about this technology and the downstream effects of introductory students learning with it.

Acknowledgments

This work was partially supported by the National Science Foundation through awards OAC-2321020, OAC-2321015, OAC-2321016, OAC-2321017, and OAC-1924023.

References

[1] J. Anderson and H. Franceschi. *Java Illuminated*. Jones & Bartlett Learning, 6th edition, 2024.

[2] J. Brown. *The World Café: Living Knowledge through Conversations that Matter*. PhD thesis, Fielding Graduate University, 2001.

[3] K. Bruce, A. Danyluk, and T. Murtagh. *Java: An eventful approach*. Prentice Hall, 2005.

[4] K.B. Bruce, A.P. Danyluk, and T.P. Murtagh. Event-driven programming is simple enough for CS1. In *Proc. 6th Ann. Conf. Innovation and technology in computer science education (ITiCSE)*, pages 1–4, 2001.

[5] Kim B. Bruce, Andrea Danyluk, and Thomas Murtagh. Introducing concurrency in CS 1. In *Proc. 41st SIGCSE Technical Symp. Computer Science Education (SIGCSE TS)*, pages 224–228, 2010.

[6] CC2020 Task Force. *Computing Curricula 2020: Paradigms for Global Computing Education*, 2021. https://www.acm.org/binaries/content/assets/education/curricula-recommendations/cc2020.pdf.

[7] A. Chtchelkanova, S. Das, C. Das, F. Dehne, M. Gouda, A. Gupta, J. Jaja, K. Kant, A. La Salle, R. LeBlanc, A. Lumsdaine, D. Padua, M. Parashar, S. Prasad, V. Prasanna, Y. Robert, A. Rosenberg, S. Sahni, B. Shirazi, A. Sussman, C. Weems, and J. Wu. NSF/IEEE-TCPP curriculum initiative on parallel and distributed computing— core topics for undergraduates. version 1, http://tcpp.cs.gsu.edu/curriculum/?q=system/files/NSF-TCPP-curriculum-version1.pdf, 2012.

[8] Nell Dale, Chip Weems, and Mark Headington. *Introduction to Java and Software Design*. Jones & Bartlett Learning, 1st edition, 2001.

[9] A. Ehlert and C. Schulte. Empirical comparison of objects-first and objects-later. In *Proc. 5th Intern. Workshop Computing Education Research (ICER)*, pages 15–26, 2009.

[10] T. Gaddis. *Starting out with Java: Early objects*. Pearson Education Inc., 6th edition, 2018.

[11] T. Gaddis. *Starting out with Java: Control Structures through Objects*. Pearson Education Inc., 8th edition, 2022.

[12] M. Guzdial. A media computation course for non-majors. In *Proc. 8th Ann. SIGCSE Conf. Innovation and Technology in Computer Science Education (ITiCSE)*, pages 104–108, 2003.

[13] M. Guzdial. Exploring hypotheses about media computation. In *Proc. 9th Ann. ACM Conf. Intern. Computing Education Research (ICER)*, pages 19–26, 2013.

[14] Joint task force on computing curricula, IEEE Computer Society and Association for Computing Machinery. *Computing Curriculum 2001: Report of the joint task force on computing curricula*, 2001. http://www.sigcse.org/cc2001/.

[15] J. Lewis and W. Loftus. *Java Software Solutions: Foundations of Program Design*. Pearson Education Inc., 10th edition, 2024.

[16] Y.D. Liang. *Introduction to Java: Programming and Data Structures*. Pearson Education Inc., 12th edition, 2020.

[17] S.J. Matthews. PDCUnplugged: A free repository of unplugged parallel & distributed computing activities. In *Proc. 10th NSF/TCPP workshop on parallel and distributed computing education (EduPar)*, 2020.

[18] W. McAllister and S.J. Fritz. *Programming fundamentals using Java*. Mercury Learning and Information, 2nd edition, 2021.

[19] Leo Porter and Daniel Zingaro. *Learn AI-Assisted Python programming*. Manning Publications Co., 2024.