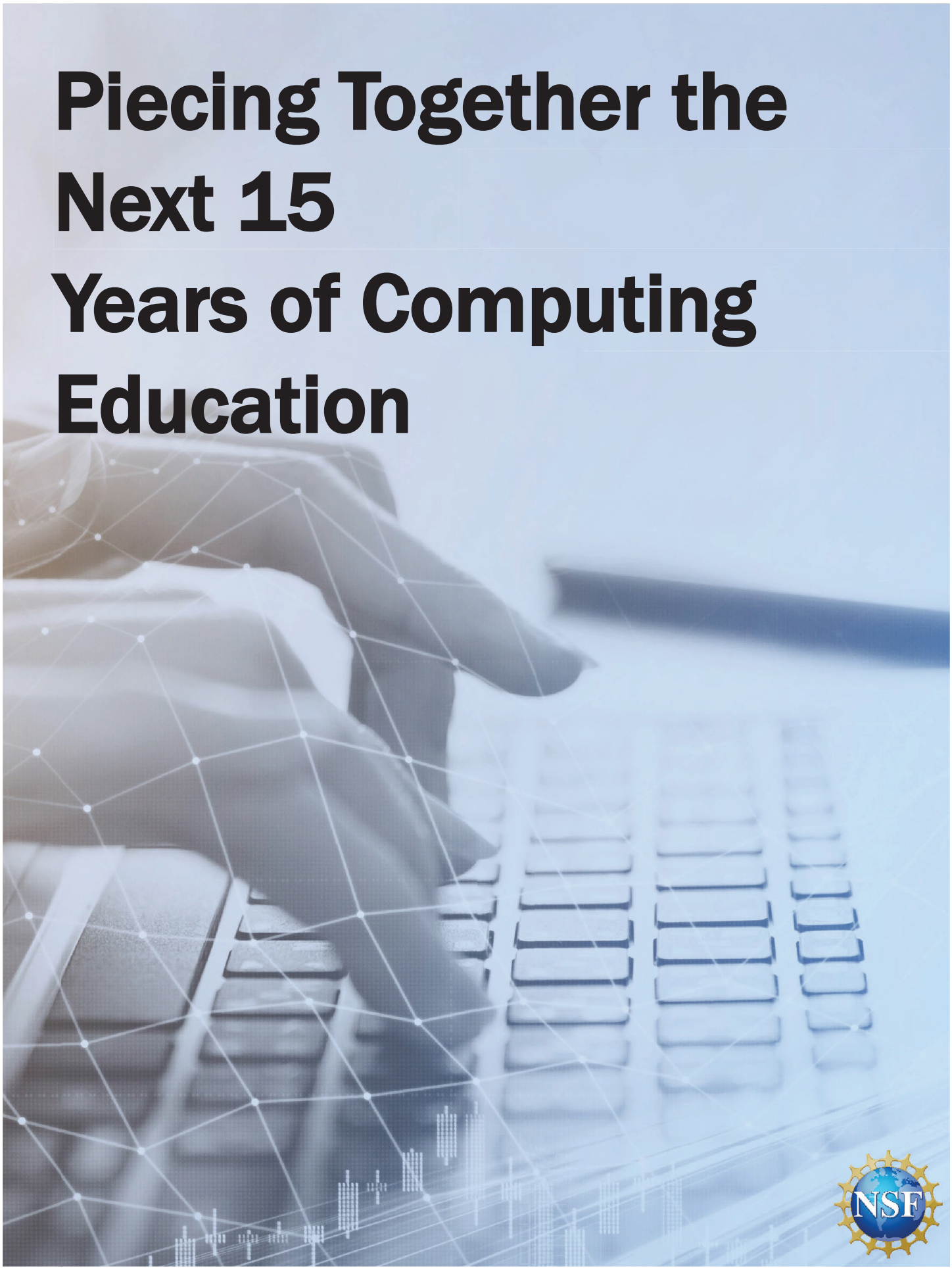
# Piecing Together the Next 15 Years of Computing Education



# Piecing Together the Next 15 Years of Computing Education

This report made possible
with support from the
NATIONAL SCIENCE FOUNDATION

Directorate for STEM Education
Division of Undergraduate Education
and the
Directorate of Computer and Information Science and Engineering
Division of Computer and Network Systems - Education and Workforce Program


This material is based upon work supported by the National Science Foundation under Grant Numbers 2039833 and 2039848.

# Piecing Together the Next 15 Years of Computing Education


*Edited by:*

Adrienne Decker, Ph.D.
*Department of Engineering Education*
*University at Buffalo*

Mark Weiss, Ph.D.
*Knight Foundation School of Computing and Information Sciences*
*Florida International University*

*Report Authors:* *(alphabetical)*

Monica M. McGill, Ed.D.
*Institute for Advancing Computing Education*

Monique Ross, Ph.D.
*Engineering Education Department*
*The Ohio State University*

Briana B. Morrison, Ph.D.
*Computer Science Department*
*University of Virginia*

David Weintrop, Ph.D.
*Department of Teaching & Learning, Policy & Leadership*
*College of Information Studies*
*University of Maryland*

Manuel A. Pérez-Quiñones, D.Sc.
*Department of Software and Information Systems*
*University of North Carolina at Charlotte*

Aman Yadav, Ph.D.
*Educational Psychology and Educational Technology*
*Michigan State University*

*Technical Writer & Editor:*
Kris O'Donnell, M.A.
*Department of Engineering Education, University at Buffalo*

## *Workshop Participants (alphabetical)*

Christine Alvarado, University of California San Diego
Austin Cory Bart, University of Delaware
Brett Becker, University College Dublin (Ireland)
Melissa Dark, DARK Enterprises, Inc.
Leigh Ann DeLyser, CSforALL
Paul Denny, The University of Auckland (New Zealand)
Brian Dorn, University of Nebraska Omaha
John Dougherty, Haverford College
Wenliang (Kevin) Du, Syracuse University
Wendy DuBow, University of Colorado at Boulder
Stephen H. Edwards, Virginia Tech
Barbara Ericson, University of Michigan
Sally Fincher, University of Kent (UK)
Kathi Fisler, Brown University
Diana Franklin, University of Chicago
Christina Gardner-McCune, University of Florida
Joanna Goode, University of Oregon
Susanne Hambrusch, Purdue University
Amy J. Ko, University of Washington, Seattle
Shriram Krishnamurthy, Brown University & Bootstrap
Deepak Kumar, Bryn Mawr College
Collen Lewis, University of Illinois at Urbana-Champaign
Xumin Lu, Rochester Institute of Technology
Susan Lord, University of San Diego
Michael Loui, University of Illinois and Purdue University
Andrew Luxton-Reilly, University of Auckland (New Zealand)
Nicholas Lytle, Georgia Tech
Lauren Margulieux, Georgia State University
Monica M. McGill, Institute for Advancing Computing Education
Briana B. Morrison, University of Virginia
Manuel A. Pérez-Quiñones, University of North Carolina at Charlotte
Christopher Proctor, University at Buffalo
Yolanda Rankin, Emory University
Monique Ross, The Ohio State University
Jean Salac, University of Washington
Simon, University of Newcastle (Australia)
Andreas Stefik, University of Nevada, Las Vegas
Claudia Szabo, Univerity of Adelaide (Australia)
Jakita O. Thomas, Auburn University
Jan Varenhold, Westfälische Wilhelms-Universität Münster (Germany)
David Weintrop, University of Maryland
Aman Yadav, Michigan State University

# Table of Contents

# Preface

This report is the result of a multi-year effort sponsored by the National Science Foundation (grant nos. 2039833 and 2039848). The original goals of the project were to bring together thought leaders in Computer Science Education, both researchers and practitioners from many different types of institutions and in different positions and points in their career to ruminate, deliberate, and postulate about what computer science education research should look like in 15 years time. COVID had other plans. While we assembled a team of over 40 leaders in the computing education space, we were unable to travel during the first two years of this effort, effectively crippling our ability to bring everyone together for a big event. We persevered and met virtually in smaller groups and created a series of small reports about research directions in subfields of computer science education. Those reports are available from the project website (https://cerfutureworkshop.wpcomstaging.com/summary-of-outcomes/).

However, what the series of reports failed to capture (due to the disjoint nature of the small group meetings) was a vision of what the field should be focusing upon. As such, we convened a meeting of a much smaller group of participants and worked with them over several months to shape a document that could be a vision of where the field should be headed. This document is the culmination of that part of the effort. The document provides challenges and vision for what

computing education could and should be. We invite those who agree or disagree with that vision to use this document to shape discussions about courses, curriculum, and degree programs at their home institutions.

For those who may be focused on computing education research, we encourage you to use this report to frame your thinking about the problems that you may like to solve in the domain of teaching and learning about computing and computer science at all levels. If you happen to be teaching a course that covers these topics, the original set of reports (https://cerfutureworkshop.wpcomstaging.com/summary-of-outcomes/) are available as PDFs and provide a nice series of briefs about the discipline and some of its open questions. We are thinking of them as discussion starters and jumping off points for students and researchers who may be interested in the field and thinking about its future. Also available on our website are a series of podcasts (https://cerfutureworkshop. wpcomstaging.com/podcasts/) that complement all of this work and give additional perspectives from workshop participants.

We would be remiss if we did not thank those who helped bring these documents to life. We owe a debt of gratitude to the original set of 43 workshop participants who joined us on Zoom during 2020 and 2021 and worked to shape the original set of reports. From there, the six contributing authors to this report ran a much bigger gauntlet when we threw down the challenge to push things further. Monica, Briana, Manuel, Monique, David, and Aman, thank you for the huge amount of time, effort, and travel you put into your contributions to this effort. It is much appreciated. We also must thank our technical editor and podcast producer and host, Kris O'Donnell. Kris was asked to take on the impossible challenge

of helping to make a pile of academic writing accessible to a wider audience. Her contributions to this effort are vast and her skills are something we are in awe of every day. We thank the National Science Foundation for their support of this project, especially Paul Tymann, Jeff Forbes, and Victor Piotrowski who thought this idea was worth funding and helped support our efforts to finish this project.

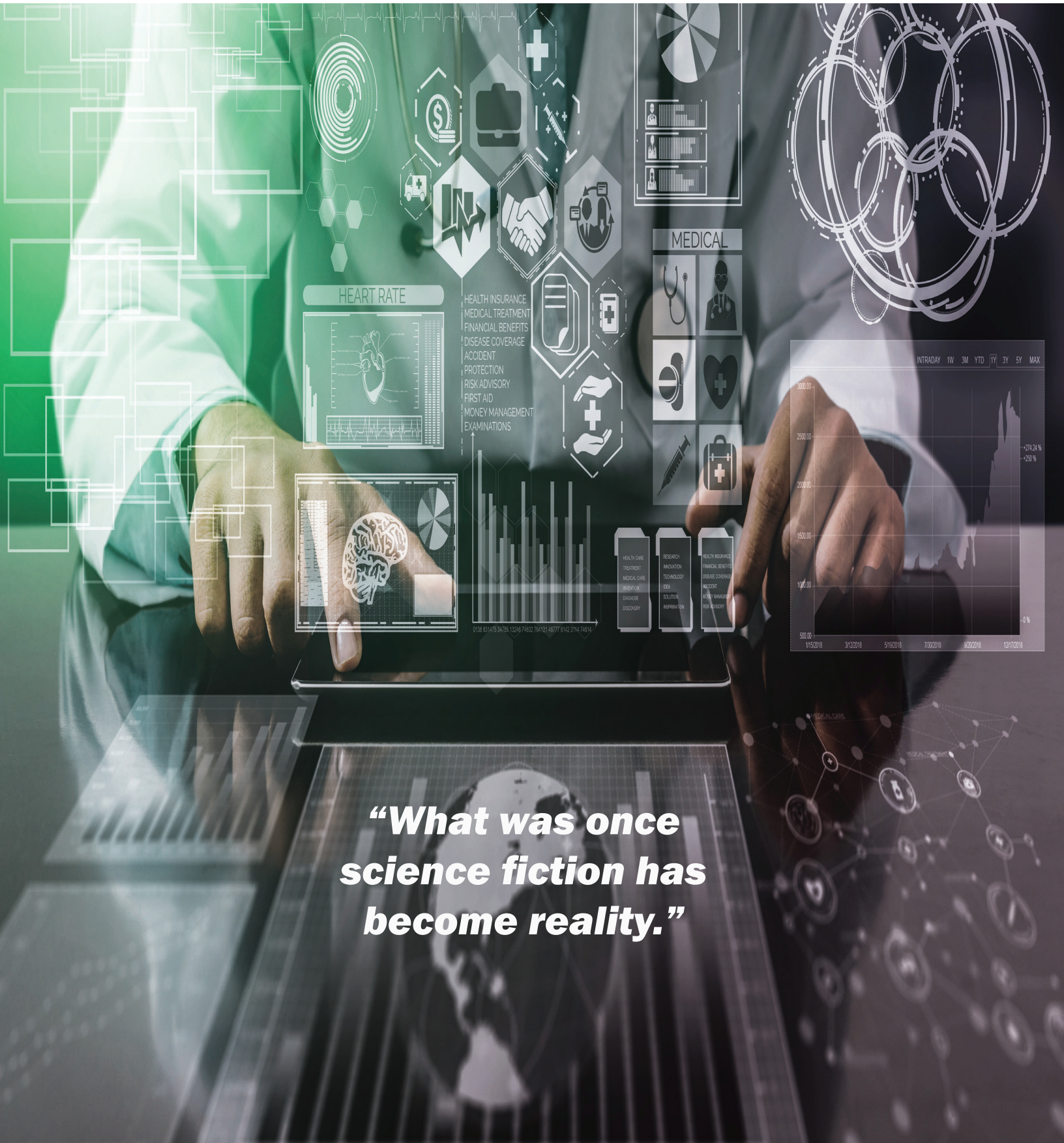
# Introduction

ChatGPT: It's a term that's barely a year old but one that is already ubiquitous. And it is Large Language Models (LLMs) like ChatGPT that could potentially have a seismic impact on education, technology, business, and society as a whole. ChatGPT and artificial intelligence are only the most recent in a string of technological advances that have revolutionized our society over the last 50 years. What was once science fiction has become reality. And this rapid pace of innovation shows no sign of abating. But in the last 50 years, one area has paradoxically remained somewhat stagnant: the way we teach and think about computing.

This raises some critical questions; how do all these advancements change the nature of work, the nature of the work of a computer scientist? And then, in turn, how does this impact educators? How do we have to adapt and change to these new demands? And how do these realities impact computing educators?

This report attempts to answer these questions. We do look at the current curriculum, one which hasn't changed dramatically in decades as a starting point for the discussion. We need to better understand how students learn computing, how students thrive in a computing classroom and in computing environments in the workplace. We also need to

understand how our marginalization of some students impacts both them and the field, and ways to bring those who are historically excluded into the field in meaningful and impactful ways.



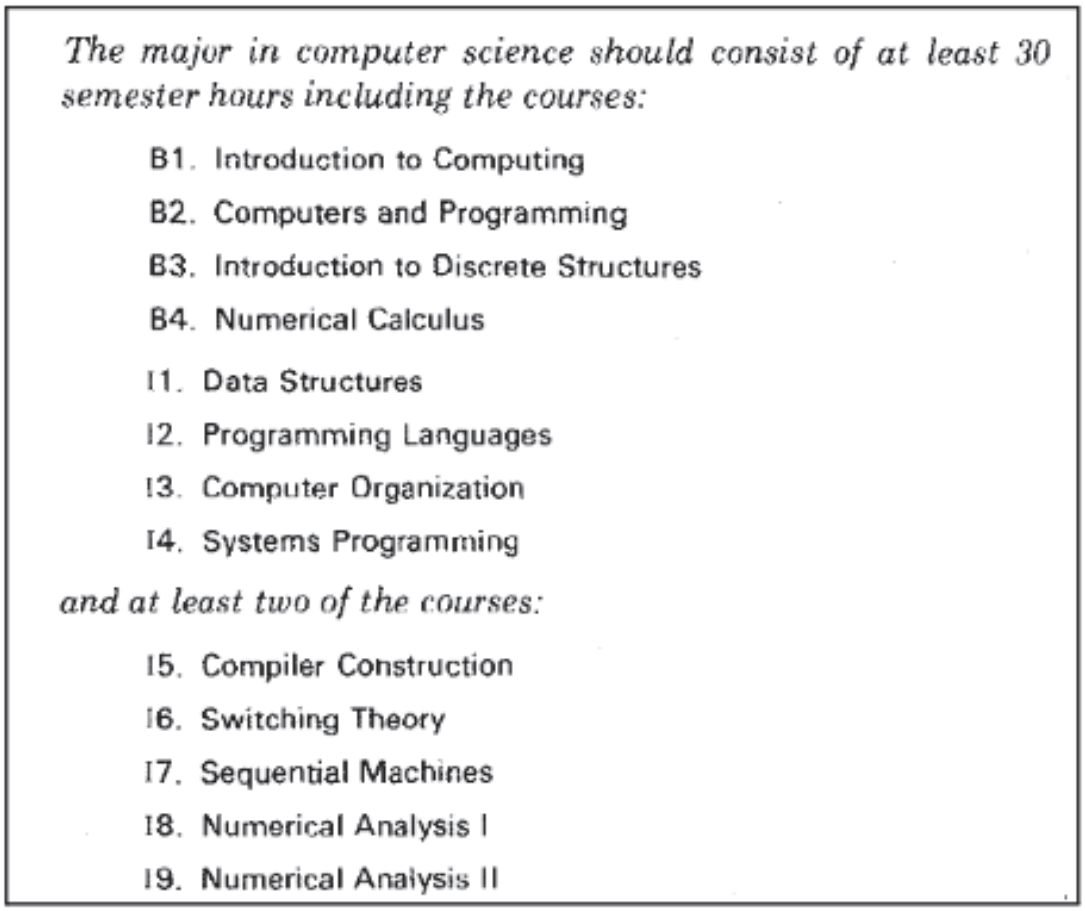
# Chapter 1: Evolution (or lack thereof) of Computer Science Curricula

There are governing bodies both inside and outside of the computing community that can significantly influence the number of credits, content areas, and outcomes for computing programs. At times, those requirements can act as a barrier to innovation. The reports and resulting curricula from these governing bodies often reflect the tension of adding new topics while retaining legacy topics; usually resulting in an overstuffed curriculum. Unfortunately, this overstuffed curriculum acts as a barrier to broadening participation (i.e., lack of entry points beyond the traditional exclusive path). With each curriculum iteration there is an opportunity to be creative in our approach to meeting the requirements levied by the governing bodies while also centering our desire to broaden participation in computing. There is a propensity to do things “the way we always have” but there is no doctrine that dictates that one course must come before another (in most cases) or that calculus is the only way to meet math requirements. Or that room cannot be made for electives that span outside of the department or college. To achieve this type of radical curriculum overhaul, there must be a thorough and critical examination of the curriculum for barriers to entry and graduation. Such critical examination may yield opportunities for creating a curriculum that allows for many entry points, that provides multiple pathways for those seeking the theoretical or the applied, and that can be easily coupled with other disciplines (outside of engineering). A curriculum that is truly for all that aspire to gain computational literacy.

***“With each curriculum iteration there is an opportunity to be creative in our approach to meeting the requirements levied by the governing bodies while also centering our desire to broaden participation in computing.”***

## Curriculum Guidelines

The content of current computer science degree programs has evolved from the first computer science program (Purdue, 1962). The Curriculum Committee on Computer Science was initially formed in 1962 as a subcommittee of the Education Committee of the Association for Computing Machinery (ACM). The Curriculum Committee published its first report, "An Undergraduate Program in Computer Science, Preliminary Recommendations," in the September 1965 issue of Communications of the ACM (Conte, et al., 1965). The first full ACM Curriculum Guidelines were published in 1968 (Atchison, et al., 1981) and presented the recommendations shown in Figure 1, along with eight recommended math courses.



*The major in computer science should consist of at least 30 semester hours including the courses:*

B1. Introduction to Computing
B2. Computers and Programming
B3. Introduction to Discrete Structures
B4. Numerical Calculus
I1. Data Structures
I2. Programming Languages
I3. Computer Organization
I4. Systems Programming

*and at least two of the courses:*

I5. Compiler Construction
I6. Switching Theory
I7. Sequential Machines
I8. Numerical Analysis I
I9. Numerical Analysis II

Figure 1: Recommendations from curriculum guidelines in 1968

Note the number of courses from this initial curriculum recommendation that are still a holdover five decades later. In the most current full ACM CS Curriculum guidelines (ACM/IEEE-CS, 2013), the recommendations moved from courses to a “Body of Knowledge” concept as shown in Figure 2.

Notice the continued growth in the amount of knowledge needed to cover the entire “Body of Knowledge” - from 280 credit hours in the CC2001 recommendations to 308 (165 + 143) for CC2013. (We acknowledge that not all Tier 2 hours are required to be covered, but this does not account for other elective hours either.) The most recent ACM CC 2020 report (ACM/IEEE-CS, 2021) moved the field toward considering a competency-based model, eliminating the mention of courses completely. While the newest report de-emphasizes the course model, little is known about how to achieve these competencies outside the traditional course model.

Consequently, at many institutions, the CS degree requirements have not changed much either. For example, examining core CS degree requirements from Purdue University’s undergraduate CS major in 1968 (Figure 3) when it was first established and comparing it to today (Figure 4), there are many similarities. Many degree programs still maintain a heavy emphasis on mathematical and technical computing knowledge.

To ensure that CS students develop critical perspectives about the design and implementation of technologies, we need to move away from an explicit focus on technical skill, and think critically about the skills computing professionals will need in the coming decades.

| Knowledge Area | CS2023 | | CS 2013 | | CS2008 Core | CC2001 Core |
|---|---|---|---|---|---|---|
| | CS | KA | Tier 1 | Tier 2 | | |
| AL - Algorithms and Complexity | 32 | 32 | 19 | 9 | 31 | 31 |
| AR - Architecture and Organization | 9 | 16 | 0 | 16 | 36 | 36 |
| CN - Computational Science | – | – | 1 | 0 | 0 | 0 |
| DS - Discrete Structures (CS2023: Mathematical and Statistical Foundations) | 5 | 0 | 37 | 4 | 43 | 43 |
| GV - Graphics and Visualization (CS2023: Graphics and Interactive Techniques) | 4 | 76 | 2 | 1 | 3 | 3 |
| HCI - Human-Computer Interaction | 8 | 16 | 4 | 4 | 8 | 8 |
| IAS - Information Assurance and Security (CS2023: Security) | 6 | | 3 | 6 | – | – |
| IM - Information Management (CS2023: Data Management) | 9 | 23 | 1 | 9 | 11 | 10 |
| IS - Intelligent Systems (CS2023: Artificial Intelligence) | 11 | 12 | 0 | 10 | 10 | 10 |
| NC - Networking and Communication | 7 | 24 | 3 | 7 | 15 | 15 |
| OS - Operating Systems | 8 | 8 | 4 | 11 | 18 | 18 |
| PBD - Platform-based Development (CS2023: Specialized Platform Development) | 3 | 0 | 0 | 0 | – | – |
| PD - Parallel and Distributed Computing | 9 | 26 | 5 | 10 | – | – |
| PL - Programming Languages (CS2023: Foundations of Programming Languages) | 23 | 21 | 8 | 20 | 21 | 21 |
| SDF - Software Development Fundamentals | 43 | 0 | 43 | 0 | 47 | 38 |
| SE - Software Engineering | 6 | 23 | 6 | 22 | 31 | 31 |
| SF - Systems Fundamentals | 18 | 9 | 18 | 9 | – | – |
| SP - Social Issues and Professional Practice (CS2023: Society, Ethics, and Professionalism) | 17 | 14 | 11 | 5 | 16 | 16 |
| Total Hours | **268** | **325** | **165** | **143** | **290** | **280** |

Figure 2: Hours required in curricula published in 2001, 2008, 2013, and 2023. The 2023 curriculum is still in draft at the time of this publication, but the numbers reflect the current breakdown for each knowledge area.

| Course | Computer Sciences Major | Computer Sciences Option in Mathematics | Curriculum'68 |
|---|---|---|---|
| Calculus | 3 | 3 | 3 |
| Advanced calculus | 1 | 1 | 1 |
| Linear algebra | 1 | 1 | 1 |
| Programming 1 and 2 | 2 | 2 | 2 |
| Numerical methods | 1 | 1 | 1 |
| Theory | 3 | 2 | 1 |
| Computer systems | 2 | 0 | 2 |
| Programming languages | 0 | 0 | 1 |
| Electives—Computer Sciences | 2 | 2 | 2 |
| Statistics | 1 | 0 | 0 |
| Electives—mathematics/Computer Sciences/statistics | 0 | 0 | 2 |
| **Total courses** | **16** | **12** | **16** |

*Figure 3: CS degree requirements from Purdue University in 1968 (Source: Rice & Rosen, 2002)*

| Course | Title | Credits |
|---|---|---|
| CS 18000 | Problem Solving and Object-Oriented Programming | 4 |
| CS 18200 | Foundations of Computer Science | 3 |
| CS 24000 | Programming in C | 3 |
| CS 25000 | Computer Architecture | 4 |
| CS 25100 | Data Structures and Algorithms | 3 |
| CS 25200 | Systems Programming | 4 |

*Figure 4: CS degree requirements from Purdue University in 2019 (Source: Purdue CS)*

## Specialization of Degrees in Computing

As the computing field continues to expand, the tendency has been to incorporate these new areas into existing courses, thus covering more material in the same amount of time; or adding more elective courses, giving students a shallow breadth coverage of topics rather than a deep understanding of core topics. This results in undergraduates with foundational computing knowledge, but limited experience within a single advanced topic. It also means that, beyond the foundational knowledge, faculty cannot guarantee students have any pre-requisite knowledge from other advanced topics, creating a myriad of individual courses that do not integrate across topic areas. For example, a web development class cannot necessarily guarantee that all students have had a database course and therefore may not be able to incorporate a database backend into the course. If databases become required, then it narrows the set of students who can take the course by introducing additional prerequisites or by privileging students who have had additional prior educational experiences beyond those required by the program.

One way to address the expanding number of topics has been to splinter the degree into different degree programs, each with its own specialty. Within computing, we are already seeing this occur. The most current ACM/IEEE Curriculum Guidelines, produced in 2020 (ACM/IEEE-CS, 2021) cover many different computing degree programs:

- Computer Engineering
- Computer Science

- Cybersecurity
- Information Systems
- Information Technology
- Software Engineering
- Data Science
  (formally published in 2021)

Nevertheless, Computer Science remains the most common of these academic offerings and often includes pieces of the other types of degrees in the form of technical electives, certificates, tracks, or minors. The splintering of computing into different degree programs comes with its own set of challenges. Explaining the differences to incoming university students (and their parents) between degrees and the resulting career paths for each can be difficult. Helping students navigate the choice of degree programs, whether during their first term or in their fourth or fifth term, can be challenging. Finally, and perhaps most consequentially, creating specializations can introduce additional barriers to computing fields for students from historically excluded populations in computing as it requires additional prior knowledge about the field before entering university and can introduce more consequential decisions earlier in the degree program.

In this proliferation of computing degrees, we differ from the historical development of engineering programs. Many of these computing degrees are often not in the same college. Software Engineering and Computer Engineering can be placed in the College of Engineering due to their connection

to ABET. Information Systems may be closely allied to Business schools. Colleges of Information Science (iSchools) are increasingly offering degrees in human-computer interaction, data science, and social aspects of computing and technology. Data Science, due to its strong foundations in Statistics and many applications to other disciplines, seems to be gaining some level of independence that the other computing disciplines have not had. With all of these degrees, it is not clear what connects us all under the Computing umbrella. Is it programming? Is it the practical application of computational solutions, analogous to engineering "application of science to the common purposes of life"? Are some of these disciplines a more applied version of a Computer Science degree? Is the relationship between the more applied versions and the mainstream Computer Science similar to the relationship between Engineering Technology and traditional Engineering degrees?

We have no reason to suspect that additional new topics will not continue to be discovered, which may in turn lead to even more specialized computing degree programs. The question becomes, what happens with the original Computer Science degree program? Does it continue to serve the "one-size-fits-all", "jack-of-all-trades" need? Or can it too become more specialized? But, if it doesn't specialize, will the "generic" degree become so abstract that it is irrelevant for industry-bound graduates? If so, would this generic CS degree be only relevant for the small number of students attending graduate school? With the advent of micro-credentials, certificates, and other non-academic credentials (e.g., industry sponsored courses or development programs), will these now be required for entry level computing

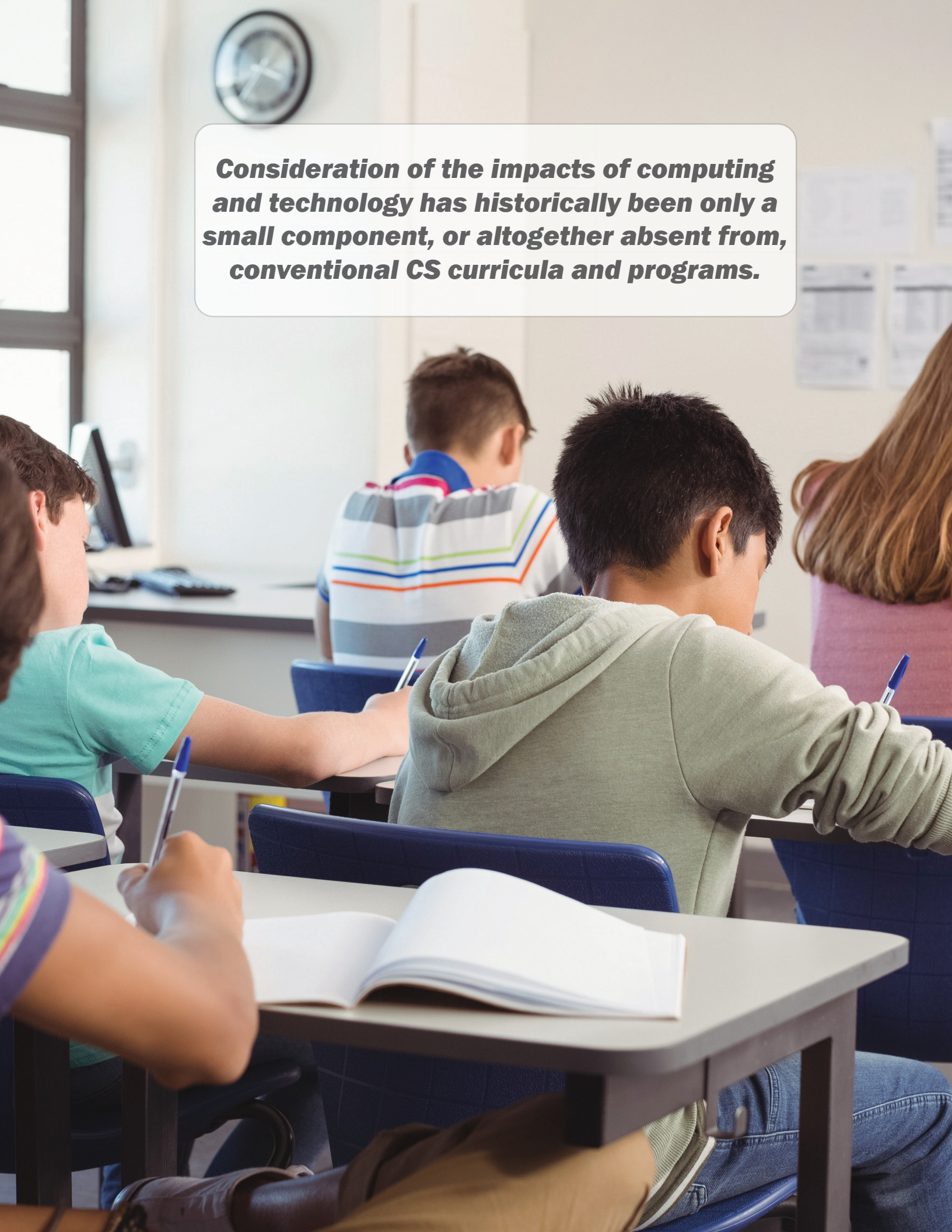
positions, even for those with generic computing degrees?

We believe that computing education and its many programs are at a decision point. The amount of required knowledge (and limited number of credit hours allowed) for a computer science degree has reached a saturation point. We see that, in general, we are preparing students for at least two distinct career paths: software developer or computer scientist. The software developer may be an applied computer scientist, using learned knowledge and skills to create and develop software artifacts for industry and society. The computer scientist is likely a more research-oriented path, with more theoretical than applied knowledge. While there is some overlap in the required knowledge for both, the skills needed for developer versus researcher are quite different and require different educational paths.

# Chapter 2: What Needs to Change?

In the last 50 years, computing, and the technologies it enables, has changed the world. This can be seen across almost all aspects of life, ranging from how we work, how we communicate, and how we relax, to how communities, organizations, and countries operate. Almost no aspect of modern society has been left untouched by the rise of computing and technology. Further, this rate of influence of computing shows no signs of slowing down. Instead, new technologies with new capabilities are continually being introduced. The rise of artificial intelligence and machine learning are changing how we think of what is possible with computing while the emergence of quantum computing sits tantalizingly on the horizon, waiting to usher in a new era of computing, computing capabilities, and a new wave of impacts on individuals, communities, and societies.

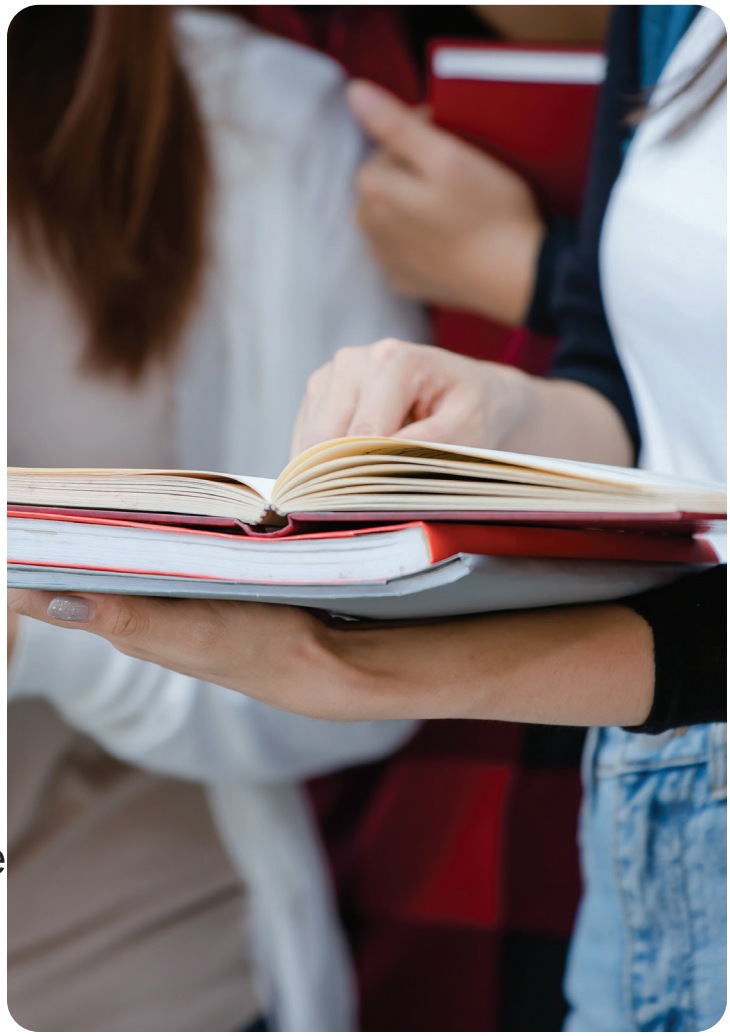
Computing is now inextricably linked with society, with the ideas and innovations emerging from the field having direct and lasting impacts on the world beyond. These influences can be subtle and nuanced or can precipitate seismic social and societal shifts. Consideration of the impacts of computing and technology has historically been only a small component, or altogether absent from, conventional Computer Science curricula and programs. However, given the increasing impact and importance of the field, Computer Science departments and programs must acknowledge this reality and take concrete steps to better prepare their students for the responsibilities that



***Consideration of the impacts of computing and technology has historically been only a small component, or altogether absent from, conventional CS curricula and programs.***

accompany the skills their students are developing.

To design and develop computing tools that do not perpetuate existing or introduce new inequities into the world, students need to understand how individuals, communities, and society operate and computing's role within and across these spheres. This understanding should include both contemporary examples as well as historical context to help students situate the impacts of computing both presently and in the future. Looking at institutions of higher education, opportunities for students to study these topics reside outside of Computer Science and other Engineering departments. Instead, learning opportunities and the associated expertise reside in departments focused on the Humanities and Social Sciences. Fields including Anthropology, Sociology, Gender and Cultural Studies, History, and Philosophy can help budding computer scientists understand the impact their craft can and does have on individuals, cultures, and the world at large. As such, making experiences that provide this larger perspective and context as part of computer science programs will help prepare well-rounded and informed computer scientists.

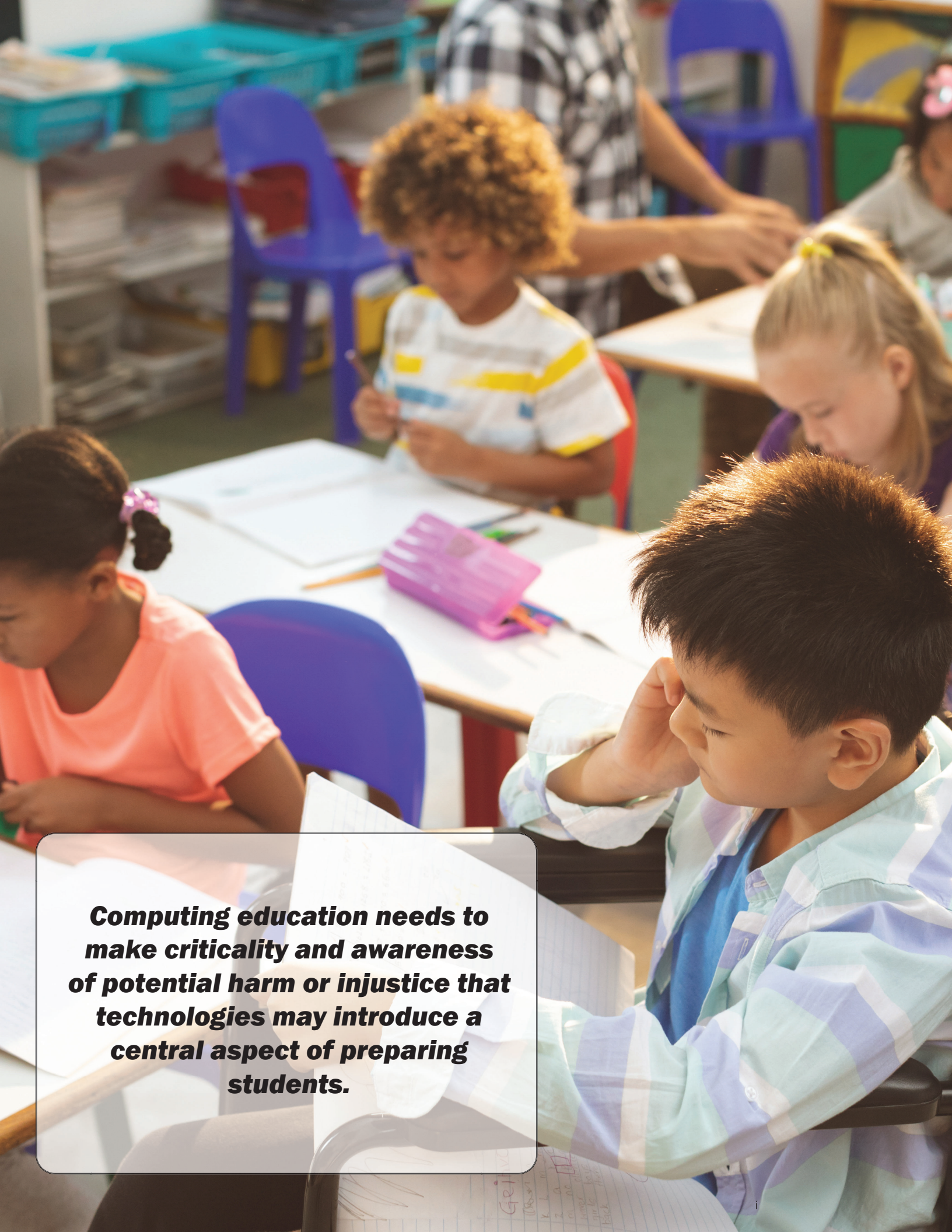


We recommend that computer science educators critically examine the curriculum and develop ways in which these types of perspectives can be integrated with the learning of technical knowledge. In the following sections, we provide more explanations and suggestions for how this could be realized.

## Preparing Ethical and Responsible Computer Scientists

We know that biased data and models based on those data can have unintended consequences. That idea should be part of computing courses and curricula and should be embedded within the context of coursework rather than being seen as a separate knowledge and skill set. Computer science educators need to incorporate socio-political aspects of technologies into the computing curriculum rather than treating these ideas as an isolated skill students need to develop.

There is a growing ethical crisis in computing where the design and deployment of technologies is causing harm, particularly towards marginalized groups. From biases in facial recognition technologies to sentencing software that recommends higher sentences for black and brown defendants, the biases and injustices that shape society are being embedded into technology. Ruha Benjamin pointed out how racialized data that gets embedded in technologies is the “New Jim Code” (Benjamin, 2019). Yadav and Heath (2022) also highlighted the invisible role computing plays in oppressing and harming individuals from marginalized groups and the need to re-examine design and implementation of technologies. Computing education needs to make criticality and awareness of potential harm or injustice that technologies may introduce a central aspect of preparing students. This includes educating them “on how anti-Blackness and racism structure the technological design process as well as use of the technologies” (Yadav & Heath, 2022, p.454). Similarly, Ko and colleagues (2020) suggested that, as part of gaining a degree in computer science, students should be taught how to tackle the myth of



***Computing education needs to make criticality and awareness of potential harm or injustice that technologies may introduce a central aspect of preparing students.***

neutral technologies and know that “software is often wrong; software always embeds its creators’ values and biases; and software can only solve some problems, and in many cases, creates new ones” (Ko et al., 2020, p. 32).

One way computing programs address this is through ethics accreditation and curriculum requirements, such as ABET accreditation standards for Computer Science programs. The ABET accreditation standards state that “the graduates of the program will have an ability to: recognize professional responsibilities and make informed judgments in computing practice based on legal and ethical principles.” Similarly, the ACM/IEEE Computing Curricula 2020 report requires professionalism and ethics as a permanent element of any computing curriculum. However, the report itself only provides a surface level discussion of what constitutes ethics within computing. As an example, the report states that professionalism and ethics in the “introductory courses in the major could include discussion and assignments on the impact of computing and the internet on society and the importance of professional practice. As students proceed in their second-year courses, they could start to keep records of their work, as a professional might, in the form of requirements, design, test documents, and project documents such as charters and project reports.” (p. 76). In practice, such a broad inclusion of ethics means that computing programs can include only a cursory nod to ethics in their program. In contrast, a more systematic integration of ethics in the CS curriculum could start with students first developing an understanding of ethics from a moral perspective within humanities and then having opportunities to consider ethical aspects of each project/assignment within their computer science coursework.

The ACM Code of Ethics is another guiding ethical framework designed to “inspire and guide the ethical conduct of all computing professionals, including current and aspiring practitioners, instructors, students, influencers, and anyone who uses computing technology in an impactful way.” (ACM, 2018). However, according to research, it is unclear how the ACM code of ethics influences software developers’ decision making, if at all (McNamara et al., 2018). This is true for both undergraduate software engineering students as well as for professional software developers and their software-related ethical decision making. Given these findings, in conjunction with technological harms, the question remains: how do we ensure computing students develop an ethical and moral compass?

An important step towards addressing this issue is to help computing students understand that technologies are political in the same way that societies are. Biases and structures that guide larger society also influence the design and deployment of technologies (Yadav and Heath, 2022). One mechanism to achieve this is to help students develop the skills to critically question and evaluate technologies before and during their design as well as dismantling them after their deployment. For example, when developing machine learning algorithms that use training data sets, students should be aware of how biases can be introduced and could potentially lead to harmful outcomes for some people. A focus on how biased data and models can have unintended consequences should be part of computing curricula.  This discussion should be embedded within the context of machine learning coursework rather than being seen as a separate knowledge and skill set.

## Broadening Students' Perspectives on the Role of Computing in Society

Most of the coursework a computer science student takes en route to a degree in computer science is focused on the intellectual and technical aspects of computing. Courses teaching programming skills, algorithm analysis, data structures, and computational theory live alongside topics like compilers, databases, human-computer interaction, artificial intelligence, and programming languages to constitute a computer science degree (see Chapter 1). Often absent, or at least underrepresented, from the slate of courses computer science students take are courses considering the impacts of computing. "Impacts" is an intentionally broader term meant to capture the various ways technology influences and shapes individuals, communities, and society. These influences can be very visible, such as the emergence of social media as a mechanism for sharing ideas and disseminating information (or misinformation) and the way smart phones reshaped the way to navigate the world. At the same time, these impacts can also be completely invisible, such as the way an individual's data is collected and sold, which can, in turn, shape the advertisements they see or the access they have to social and financial services (Noble, 2018; O'Neil, 2016).

Beyond the experiences of

individuals, consideration for how new technologies enabled by computing can upend industries and economies also has significant implications for individuals and societies. This focus on the impacts of computing can be seen in the organization of K-12 computer science instruction (K12 Computer Science Framework, 2016) but is less present as students advance to post-secondary education. Attending to impacts of computing is discussed as part of the most recent guidelines for computing curricula published by a joint ACM/IEEE task force (CC2020 Task Force, 2020); however, the recommendations for how and where these topics reside serves as only an initial step to helping computer science students understand the full set of potential impacts of computing at individual, community, and societal levels. A key mechanism for accomplishing this in higher education is to have computer science students take courses in humanities and social sciences departments.

Bringing humanities/social sciences and computing together allows students to see the limits of technologies and technology-oriented solutions to complex issues and also serve to ensure that students don't see computing as being isolated from the socio-political realities of our society. Too often computer science and technologies are seen as solutions to social and political issues. Consider the number of times techno-solutionist approaches are suggested to tackle school gun violence from metal detectors to AI-based taser drones even as techno-solutions have been shown to continually fail (Heath and Yadav, 2022). As such, it is important for students to understand that computing and technologies can't always provide solutions to address socio-political problems. To do this, it is essential that computer science students learn from and with humanities/social

***Bringing humanities/social sciences and computing together allows students to see the limits of technologies and technology-oriented solutions to complex issues and also serve to ensure that students don't see computing as being isolated from the socio-political realities of our society.***

sciences faculty, who can lend their expertise to co-develop curricula and courses at the intersection of humanities and computing.

Providing computer science students opportunities to understand the impacts that computing can and does have on society is important given the role that these students will play in creating and shaping the next wave of tools and technologies. While computer science departments can, and often do, offer a course attending to the impacts of computing, sometimes as part of a larger professional practices course, these courses are only a first step for helping students understand the impacts of computing. Looking beyond Computer Science departments, colleges of Information Studies, Communications departments, or interdisciplinary campus entities with names like “Technology, Society, and Behavior” or “Computing for the Arts and Sciences”, provide classes focused on sociotechnical systems, directly attending to the intersection of technology and individuals or societies. Looking further afield, courses in social sciences and humanities departments, while often not directly related to computing or technology, can lay the conceptual foundations for helping computer science students understand the impacts of computing on individuals, communities, cultures, or countries. Providing opportunities for students to take courses from outside the computer science department will prepare them to more fully understand the potential impacts of computing and their role in it as they move from academia into the world. It also exposes them to differing worldviews and ways of knowing to aid in the development of critical thinking beyond that in technical fields.

## Developing the Ability to Take Multiple Perspectives

It is essential that students graduating from computer science programs have the ability to understand and value the perspectives of individuals from various backgrounds and with differing life experiences. This includes perspectives of individuals from different cultures, nationalities, socio-economic classes, races, gender identities, religions, political orientations, physical and neurological abilities, and intersections across these and additional groups. This is not to say graduating computer science students need first-hand experiences across all these groups and intersectionalities but that students need to develop the skills to learn from, empathize with, and make informed decisions that consider the perspectives of and value the contributions of individuals from these groups. This perspective-taking ability is essential as the technologies and tools developed by students graduating from computer science programs will be used by and impact individuals from across all these groups. So, it is imperative that graduating students have the ability to consider their needs and desires,can think through and consider the implications their work may have for them, and can value the knowledge and contributions of these groups

Developing these abilities comes through both educational and lived experiences. By this we mean these skills develop through both intellectual exercises, such as taking courses focused on exploring the lived experiences of others or theories and ideas that can describe them, and by having first-hand experience working closely with individuals with a different perspective and/or set of lived experiences.

## Working with and for Those Outside of Computing

Another productive outcome of having computer science students take courses from the humanities and social sciences stems not only from the intellectual growth that will emerge from these courses but also from the social experiences that will result. Enrolling in courses from fields such as Anthropology, Philosophy, or History, computer science students will end up sitting alongside and collaborating with students who may know very little about computer science. It is also likely that in taking these courses, students will work closely with students who have different educational backgrounds, prior experiences with computing and technology, and distinct cultural resources they bring to the shared tasks.

These experiences of working closely with students from outside of computer science will have numerous benefits. One benefit of these collaborations is it will help computer science students understand the value of multiple perspectives on a project. In particular, students will learn the value of ideas and input from those outside the computing discipline and from individuals from groups historically excluded from computing and STEM disciplines. Recognizing that computer science programs are disproportionately populated by White or Asian males from high socio-economic backgrounds, these non-computer science collaborations may serve as essential opportunities for computer science students to collaborate with students with very different sets of cultural resources and prior experiences. In seeing a peer contribute essential ideas and expertise unrelated to computing and/or very distinct from their own experiences and contributions, a computer science student will gain firsthand experience of

working on a project where they themselves would not have the knowledge and experience to complete the project on their own. In this way, the experience of group work and collaboration is not merely an exercise in the distribution of work (i.e. the project can be completed more quickly as the work is separate) but instead, students will experience a project where the result is fundamentally different due to differing perspectives and inputs. The result is being able to see how collaborating with students with different perspectives and ideas can shape the resulting product and the final results of a collaboration will be something that no one student could have accomplished alone.

A second valuable benefit to working closely with students from outside computer science is that it will provide valuable firsthand experience communicating with and working alongside those who may not have the same level of computational or technology expertise. In working closely with those without the same level of computing expertise, it will help illuminate emerging expert blindness that computer science students develop by only working with other computer science students. The idea of expert blind spots comes from the education literature and attends to the idea that as expertise develops, individuals forget what it is like to not know something, which in turn, impacts how they think and talk about an idea (Nathan & Petrosino, 2003). In the context of a computer science

student working with non-computer science students, this may take the form of the computer science student assuming their partner understands the idea of “versioning” when working on a draft or the expectation that they can easily shift between different software tools/platforms as part of their shared work. This may also emerge in social and collaborative aspects of the project, where a computer science student might expect their non-computer science peers to use the same time management tools or communication and collaboration platforms (i.e., use Slack or Discord rather than WhatsApp or email). Experiences collaborating and communicating with non-computer science students will prove valuable as communication is an essential skill in the workplace.

Related to this last point, close collaboration with non-computer science students will help prepare computer science graduates for the professional world. While some computer science students may graduate and find their future careers are largely spent surrounded by other computer scientists (e.g., graduate school, research-focused positions), many students will end up in professional contexts where they are regularly interacting with and communicating with those from outside computing. For example, a computer science graduate that takes a position working in the private sector, be it a Fortune 500 company or a start-up, will end up frequently teaming with individuals with different backgrounds, including designers, salespeople, and colleagues working in product development, customer relations, or advertising. Having educational experiences communicating with and collaborating on projects with individuals with these differing backgrounds will prove beneficial.

A final, key benefit of collaborating with students from outside of computer science is that it will help make concrete the idea of designing computing tools for non-computer scientists. This idea stems from the ideas of creating personas as part of the design process. Personas come from the field of human-computer interaction and design and are used as a mechanism for representing and considering end-user needs by creating realistic portraits of potential users (Miaskiewicz & Kozar, 2011). Research has found that personas serve as productive tools for supporting designs for others. Having computer science students work closely with non-computer science students, these collaborators can serve as specific, concrete examples of users that computer science students are building tools for. Through these collaborations, computer science students may gain firsthand experience and perspective on how non-computer science students use computing to accomplish tasks. That experience can serve as a generative reference point for considering who is using the tools being designed, what challenges they may face and the support they may need. This experience can help put a face on the idea that tools developed by computer scientists are often not used by computer scientists, and thus, may not always be used in exactly the way they are intended. Having computer science students take courses outside of computer science and collaborate closely with non-computer science students can help future computer scientists and software developers understand this and help inform their future decisions in response to it.

## Address Equity and Social Justice in Computing and Computing Education

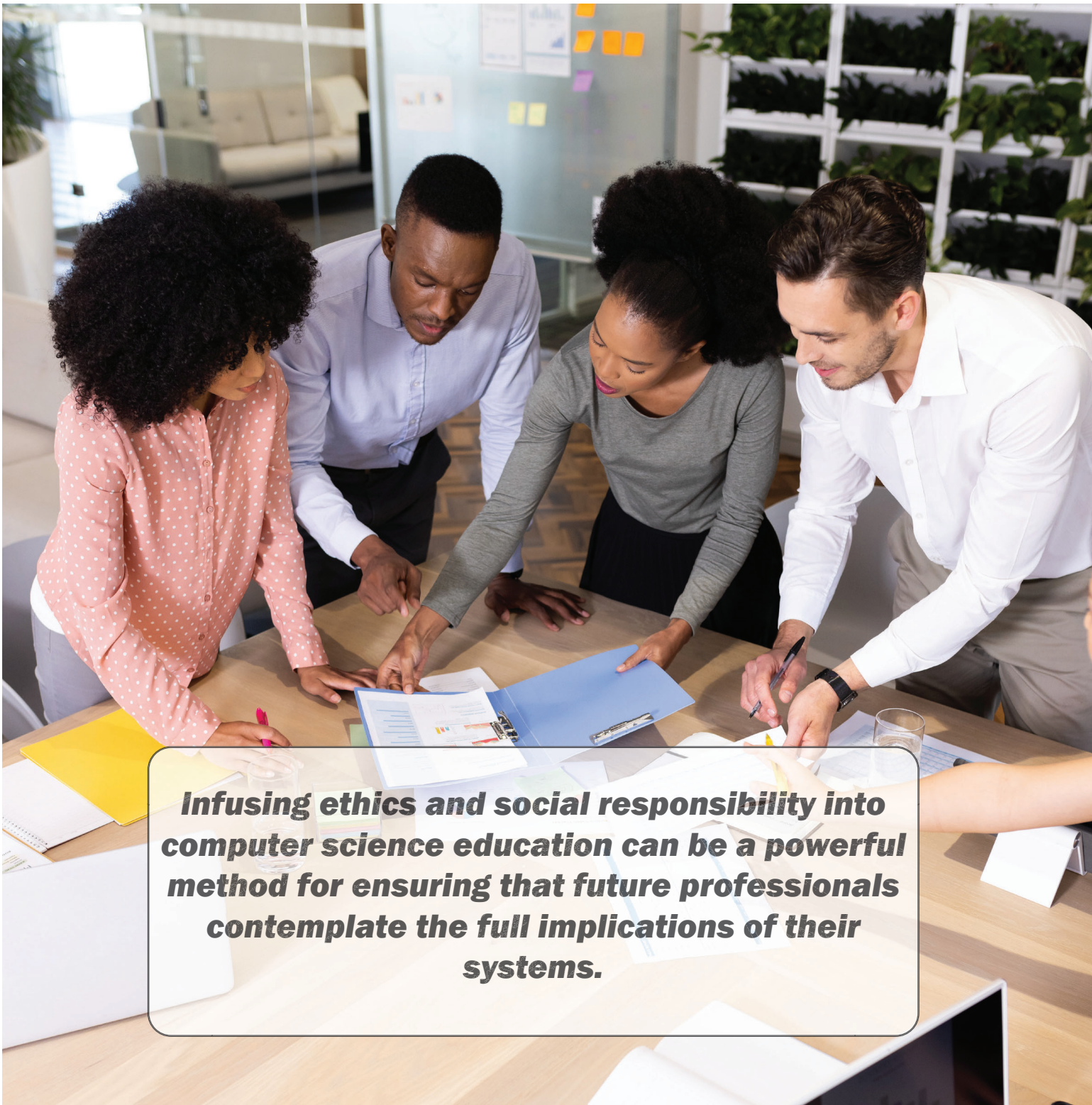
In tech, the focus is often on how quickly innovations can be disseminated. Therefore, little attention is generally paid to the repercussions and detrimental impacts that they may have on society, communities, and individuals. Technological change produces results that are a trade-off between what they give and also what they take away.

Systems supported by computing are often driven by corporations and other institutions that may lack the motivation and, therefore, the will to adequately consider and respond to the ethics and social responsibility in the systems that they create and sustain. This has already led to additional marginalization of groups who are not part of the computing power structure and primarily reflect the values that the creators have enthusiastically embraced and upheld (Kaspar, et al, 2023).

Additionally, radical shifts in computing are being driven by the growth of artificial intelligence and machine learning, and these radical shifts are expected to continue. These shifts may be as significant as the Gutenberg press and will require inspection into how they contribute to equity and inequities, justice, and injustice.

Infusing ethics and social responsibility into computer science education can be a powerful method for ensuring that future professionals contemplate the full implications of their systems. Despite this call for action having been raised decades before in computing education (Granger, et al, 1997), it remains wholly unrealized. Over the next 15 years,

computer science education must consider how technology contributes to the further stratification and marginalization of communities and individuals, especially those who are not part of the technological change. Further, it must ensure that there are sufficient learning opportunities for all citizens to be aware of how they as an individual and their communities are impacted by technology.



***Infusing ethics and social responsibility into computer science education can be a powerful method for ensuring that future professionals contemplate the full implications of their systems.***

# Chapter 3: How do we Change the Curriculum?

The role computer science plays in the design of technologies that lead to disproportionate harm towards marginalized communities requires that students understand unintended consequences of their work and develop an ethical framework that provides them with the tools to question technologies and even dismantle them if needed. This means that ideas, perspectives, and frameworks from the humanities/social sciences need to be part of core computer science curricula and not just something that is left for the margins of a computing degree program. In short, bringing humanities and computing together within undergraduate computer science degrees requires a re-shift away from just developing technical skills to developing critics (Waters, 2021).

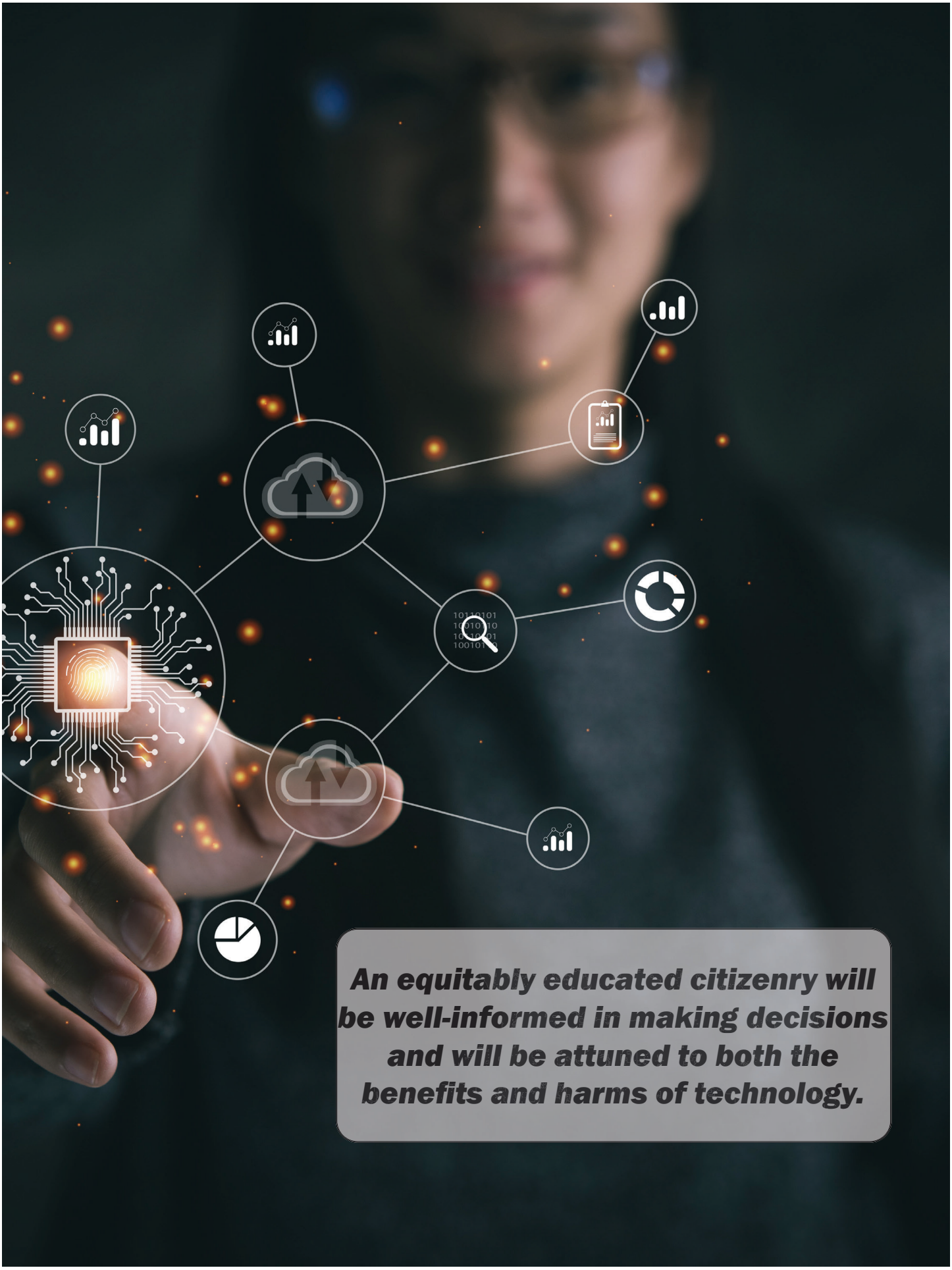
Aspirations of being an equitable and just discipline that serves humanity can only be achieved with intentionality. Equity and justice are a deliberate and distributed effort towards serving all through socio-technical solutions and to providing opportunities for educating the citizenry. An equitably educated citizenry will be well-informed in making decisions and will be attuned to both the benefits and harms of technology.

While there are multiple definitions of equity as well as how the definition is operationalized, we adopt the following definition: "...the guarantee of fair treatment, access, opportunity, and advancement while at the same time striving to identify and eliminate barriers that have prevented the full participation of some groups. The principle of equity acknowledges that there are historically underserved and underrepresented populations and that fairness regarding

these unbalanced conditions is needed to assist equality in the provision of effective opportunities to all groups." (Seramount.com, 2020, online) Bravemen and Gruskin (2003) define equity as "...an ethical concept, grounded in principles of distributive justice" and move deeper into how equity can be operationalized within a given field. Here, we provide an operationalization of equity for computing education that is a modified version of their operationalization of equity for health:

*Equity in computing education can be defined as the absence of systematic disparities in computing education (or in the major social determinants of computing) between social groups who have different levels of underlying social advantage/disadvantage—that is, different positions in a social hierarchy. Inequities in computing education systematically put groups of people who are already socially disadvantaged (for example, by virtue of being poor, female, and/or members of a disenfranchised racial, ethnic, or religious group) at further disadvantage with respect to their lives; it is essential to wellbeing and to overcoming other effects of social disadvantage.*

In the context of technology, justice can be viewed as economic, political, and social. When considering the words of Ruha Benjamin in Race After Technology, "Justice, in this sense, is not a static value but an ongoing methodology that can and should be incorporated into tech design. For this reason, too, it is vital that people engaged in tech development partner with those who do important sociocultural work honing narrative tools through the arts, humanities, and social justice organizing." (Benjamin, 2019, p. 193). We extend this definition to include those who are


***An equitably educated citizenry will be well-informed in making decisions and will be attuned to both the benefits and harms of technology.***

impacted by technology as well, who also deserve the equal right to be educated about how algorithms and the resultant technologies work and how they impact their everyday lives.

An equitable and just vision of computing education must consider:

- What should be taught, including socially responsible and ethical content,
- Who should be taught and who should be teaching, including barriers in place that must be removed for anyone to learn and to teach, and
- How it should be taught, including curriculum and pedagogy that meets the needs of all learners.

This vision directly aligns with the National Science Foundation's (NSF) mission statement, which is "to advance the national health, prosperity, and welfare; to secure the national defense; and for other purposes" (National Science Foundation, 2018, p. 5). Our nation's best approach to combat inequities and injustices are through strong computing education initiatives for all citizens, and these educational aspects to address this need directly align with the NSF's focus on societal impacts and the STEM workforce.

There are a multitude of well-documented barriers that stand in the way of the United States achieving a citizenry educated to identify and call out benefits and harms that technology has on individuals and society. Some of these include:

- Inequitable preK-12 educational system that is inextricably linked to families' income levels as it is funded based on local property taxes.
- Higher education that is increasingly economically unattainable for many citizens

- Increasing usage of technology to spread misinformation about individuals, groups, and societies, with greater spread of hate against marginalized groups.
- Increasing usage of technology to spread misinformation about the sciences.
- Increasing wealth gap.
- Increasing destabilization of our democracy.

To address these barriers, we present key aspects for computer science education that must be considered over the next 15 years. This includes challenging the idea that technology is neutral, addressing equity and social justice in computing and computing education, understanding that computing is a fundamental right, and addressing the needs for effective and equity-focused formal and informal education.

## Challenge the Idea that Technology is Neutral

Technology has never been neutral. The division between those who wield technological power and those that are more likely to be subjected to its failings has only grown in recent years, and the slowly growing proliferation of artificial intelligence, machine learning, and data science has only exacerbated this division (Postman, 1998). Yet, while there is ample evidence of the harms that rapidly evolving technological change has already caused on those outside of the power structure, computer science education largely ignores the lack of neutrality of technology and the harms that it can lead to, which only further disadvantages a vast swath of communities and individuals.

Over the next 15 years, computer science education must provide time and space for students to investigate critical questions about the lack of technological neutrality and explore its implications within the context of their learning.

## *History of Engineering in Higher Education*

Given the tensions computer science departments are facing, we can look to other disciplines to see how they have resolved similar challenges. One obvious discipline to examine is that of Engineering.

Academic programs offering engineering go back to early years of education in academia in the United States, and even then, were heavily influenced by educational frameworks from Europe. New programs emerged as needs arose in society, e.g., space exploration created the need for aerospace engineering, industrial revolution increased the need for civil engineering, the 2nd industrial revolution influenced development of electrical engineering, etc. Even the content of the curriculum and the length of the degree varied over history. Early degrees were seen as too applied to be part of a four-year degree. At one point, engineering curricula integrated humanities as a critical component of the field and was an integral part of the growth of engineering

as an academic field. Eventually, they faced the same issue computing is facing, a bloated curriculum that forced changes in curriculum designs and goals, moving many of the required humanities out. Somewhat similar to computing curricula, engineering curriculum design was largely designed and created by professional organizations. In the last few decades, there has been a shift in emphasis from science to design (e.g., practice). Recent developments include the growth in Engineering Technology, a more applied four-year degree than the traditional Engineering degrees. Both engineering technology and engineering degrees are accredited by ABET with Engineering Technology being more directly tied to the practice of engineering, using current tools and practices.

Computing may need to look further afield from engineering to other disciplines for ideas. Many have compared computing to architecture. Just as architecture is related to construction, computing is related to software development. One involves design, apprenticeship, and critiques through shared values, the other involves problem solving due to physical, monetary, and resource limitations. Perhaps we should explore the medical education model - a generalized degree followed by apprenticeship in a specialized area.

## Joint Humanities/Social Sciences and Computing Courses

One mechanism for engaging computer science students with content from humanities/social sciences is to develop and offer joint humanities/social sciences courses. These courses would be cross listed between computing and humanities/social science departments and cover topics at the intersection of the two fields. For example, courses on

topics such as Computational Journalism, Sociotechnical Systems & Design, Democracy in the Computing Age, Social Data Science, Ethical Foundations of Computing, Communications & Computing, and Technology & Human Development, explore topics at the intersection of computing and humanities/social sciences. When developing these types of joint courses, it is important that they serve both humanities/social science goals and computer science goals rather than prioritizing pushing computing concepts into other disciplines. Computing concepts should be used to enhance disciplinary teaching and learning goals within humanities. Identifying content and developing courses that are mutually supportive is a key characteristic of these types of joint courses. In taking these jointly designed and offered courses, computer science students would be given first-hand experience exploring how concepts and practices from computing can be applied outside of a stand-alone computer science  context. The cross listing of courses also means these courses can attract students from and other disciplines to work and learn together, providing valuable experiences, as discussed previously in Chapter 2.

An effective approach for creating joint humanities/social sciences and humanities courses is through co-teaching. Recruiting two faculty members, one from computer science and the second from the partnering discipline would leverage both sets of background and knowledge. This co-teaching model can lighten the load for one faculty member creating a brand-new course on their own while also making it possible to cross-list courses between two departments that do not have faculty crosslists. This greatly expands the list of possible cross listed courses as it will not rely on having a single faculty member with expertise in both areas. While

some would argue that the recent push for CS+X programs addresses this need, this approach falls short as it prioritizes computer science as a mechanism to contribute to and advance other fields rather than considering the intellectual contribution those fields can make to the learning experience of a computer science student. Additionally, the “X” in CS + “X” is often focused on STEM fields where there are opportunities to advance the discipline with computing power.

Another way to bring humanities and computer science together is to revamp degree requirements in computer science to encourage students to take courses from non-STEM disciplines that would allow computer science students to expand their understanding of social, political, historical, and economical issues. This strategy takes advantage of the fact that useful and impactful humanities and social science courses already exist on campus for computer science students to take. It also has the benefit of not requiring the computer science department to create and offer new courses. This is important given that many computer science departments are already struggling with rapidly growing enrollments and difficulties in managing

course sizes. Another benefit of this approach is that it provides maximum flexibility to students to pursue courses they are interested in from across campus. A result of this flexibility will be that students in upper-level computer science courses will end up collaborating with their peers who have taken different humanities/social sciences courses. This means a student that took a course in the communications department can work alongside students who have learned theories from sociology, philosophy, or anthropology, thus further expanding their knowledge of humanities and social sciences.

# Chapter 4: Beyond Changing the Curriculum

Given the impact of technology on society, equity and social justice need to be central to how we engage in formal education spaces. Aspirations of an equitable discipline require deliberate and distributed attention to curriculum (explicit and hidden), recruitment, retention, and pedagogical considerations in computing education.

Many of the pedagogical practices used in classrooms today were also used in classrooms 40 years ago. How many courses still revolve mainly around two or three lectures per week with programming assignments and two or three exams for assessment? Missing from this approach is the adoption of research-based practices such as peer instruction, pair programming, and mastery learning. What is required to advance the curriculum design and instructional practices of the discipline?

We claim that programming assignments haven't changed much, but the infrastructure around programming assignments has changed significantly (e.g., open source, source code repositories, automated graders). But the nature of assignments (e.g., large programming problems to be completed in isolation) have not changed much. With the disappearance of the computer lab, we no longer have communities of practice among students, as they often work alone on their personal laptops. The collaboration among students now happens online and is often limited by who you already know. Even the practice of use-your-own equipment has become an issue as we have extended the required infrastructure to include continuous wireless access. Finally, our curriculum largely ignores prior knowledge, giving an

advantage to those students that come from more privileged backgrounds. And now, with the proliferation of LLMs (Large Language Models), many of the tried-and-true problems that have historically been assigned in courses can be solved by the LLMs in a matter of seconds.

The way computing education is taught is often driven by the values and beliefs that the instructor holds, which can further exacerbate exclusivity. Pedagogical practices must be reflective of quality needed to enable students to succeed while simultaneously being aware and open to the many ways in which students learn, which is often linked to their lived experiences.

## Embracing Technological Advances for Applied Knowledge

As is evident in the slow adoption of research-based pedagogical practices within the computing classroom, computing instructors are often slow to embrace technological advances. This is often because adopting new pedagogical practices or embedding new technological products requires re-working all or a large portion of the course design. Want to try peer-instruction? Great, re-work all your presentation materials to include questions (which you have to develop along with appropriate distractors). Then find a tool to allow you to track answers from students and incorporate that into the lectures. Do you want to try using an auto-grader for your assignments? Great, pick one and then start coding up your test suites for all your assignments - after setting up the development environment. With faculty already over-committed and dealing with burgeoning enrollments, reworking the major components of an existing

class without additional time isn't a reasonable request. And while most instructors do work on class preparation during breaks, this usually involves upgrading to the most recent interpreter / compiler / textbook version and ensuring all the code examples still work as intended. The point here is that the underlying tools within computing (operating systems, compilers, interpreters, IDEs, etc.) are always being updated and changing, requiring significant effort just to keep up and maintain the status quo each semester.

Incorporating new pedagogical or technological solutions into a course requires an even larger amount of time: it's basically equivalent to creatinga brand new course. However, with the emergence of generative AI and its applications to programming and software development more generally, all computing instructors need to begin to rethink what they teach and how they assess knowledge and skills in their courses. With the advancement of tools and technologies, some topics morph from being core knowledge into research topics. Consider the concept of creating a simple website, or single webpage to be displayed through a browser. Initially, writing HTML with text editors was taught as a way to create the desired result (and of course, every computing major worth their salt should be able to create a web page!). Then came tools to simplify the

website creation process (DreamWeaver, FrontPage). Now, there are companies built on the ability to create entire websites through drag and drop, WYSIWYG interfaces (Wix, WordPress, etc.). The knowledge and skills to create a website have changed dramatically.

Does anyone still have an HTML course or unit in one of their courses? No - because the tools have made the task so easy that it doesn't require specialized knowledge. We may be reaching that point with other topics - programming languages, compilers, perhaps even programming.

Thirty years ago, it was probable that some percentage of computing majors would eventually work on writing operating systems, compilers, or other system-based software. Today, few, if any of our computing majors (with a bachelor's degree) will develop a new programming language or compiler. Operating systems and compiler construction are research topics for computing faculty and graduate students. Yet, how many degree programs still require all computer science majors to take a class in operating systems or compilers where students build an operating system or compiler? Similarly, we see specific areas of programming languages, like comparing the implementations of static and dynamic typing across programming languages as small, nuanced differences with obscure languages that most software developers will never encounter.

As the tools advance, the content of curriculum and specific courses also need to change. Just as we likely don't teach HTML anymore, soon we may not need to teach for-loops or if-statements. Generative AI tools have the potential to completely change what and how we teach introductory

***Generative AI tools have the potential to completely change what and how we teach introductory programming as much as they may change how students learn to write essays.***

programming as much as they may change how students learn to write essays. Just as the calculator changed whether students should memorize math facts, how the computer keyboard changed whether students should learn to write in cursive, and search engines changed the need to understand how a dictionary is ordered, generative AI tools may forever alter what students need to know about programming. It will be far more important to be able to construct the most appropriate prompt and evaluate the result for correctness rather than develop an algorithm from scratch. If, and how we adjust our teaching and course / curriculum content to incorporate these and future tools will define the future of the discipline.

## Retention Efforts/Supports with the Curriculum and Pedagogical Practices

It is important to have a Computing student population that reflects the full breadth of society, welcoming students from differing backgrounds and with diverse ranges of prior life and cultural experiences. This means that active recruitment and support of students who have historically been excluded from formal computer science educational contexts is an essential component of any effort to help prepare computing students for responsible professional conduct after graduation.

The complementary challenge to recruitment is retention. Retention in formal computing education spaces is a complex systems problem with many factors that contribute to a student's retention. However, we suggest there are two spaces that can have high impact towards retention: pedagogical practices in the classroom and structures and supports.

Pedagogical innovation is one way to retain students in the field of computing. Pedagogical innovation may include the adoption of evidence-based practices on a small scale or even complete overhaul of course designs. This transformation should be led by first ***evaluating computing education with the same attention to equity and social justice as conformance to curriculum accreditation***. We know that past (and in some cases current) pedagogical practices (e.g., delivery and assessment) are full of known inequitable practices - from few assessments (resulting in high stakes exams) to grading on a curve. These approaches to assessing learning are not effective and are instead widening the gap in achievement in computing. To address these inequities, exploring evidence-based pedagogical practices from other fields is a necessity. While there may not be a direct transfer of practices from one field to another, we must explore approaches like culturally relevant pedagogy to inform the work we do in computing education.

Given the unique space that computing education inhabits, we must critically examine the practices so prevalent in the discipline for opportunities to be more mindful of how we present material and the ways in which our assessment methods are misaligned with our outcomes and objectives as a discipline. The adoption of new and innovative pedagogical practices has the potential to benefit all computing students and could yield higher retention rates in the field. When considering the boom in enrollments that many institutions are grappling with, this becomes complicated, but should not be ignored. We must work to ensure, in the case of enormous class sizes, ***we do not neglect to keep equity at the forefront of designing and executing pedagogical practices in the classroom.***

Structure and support are other means of increasing retention in computing. Structure and support in this context pertain to both faculty and students. To achieve the ambitious goals related to pedagogical transformation, faculty will need professional development support, teaching support (graduate teaching assistants, undergraduate teaching assistants, instructional designers), community support, and administrative support (time, space, and recognition for innovation). Exploring ways to create communities of practice for faculty to navigate

transformation in the classroom is a form of support for faculty. Likewise, reimagining support for students that is not limited to “office hours” is also critical to retention. The development of communities of practice both inside and outside of the classroom helps with situated learning, sense of belonging, and development as professionals in the field of computing. Likewise, investigating the utility of creating communities of practice or spaces that center minoritized populations in computing might aid in retention efforts.

## Hidden Curriculum

Critical examination of computing should not stop at the explicit curriculum (courses, outcomes, sequences) but must also include the hidden curriculum. The term hidden curriculum refers to the implicit academic, social, and cultural messages, unwritten rules and unspoken expectations, and unofficial norms, behaviors and values that are transmitted to students in the context in which all teaching and learning is situated. This starts with the competition inherent to admission caps at universities that signal a culture of competition and is underscored by assessment techniques like bell curves. This includes the glorification of the myopic computer science participation - this notion that students must eat, sleep, and breathe computing to be successful. These hidden messages, norms, and values can be barriers to engagement for anyone who does not wish to engage in competition as a means of achieving their occupational pursuits.

In addition to the hidden curriculum, there are the ambient cues that make up the computing environment, i.e., pictures on the walls of computing heroes that are not inclusive (mostly men) and material we use that only leverages traditionally masculine disciplines or context (e.g., fantasy football, Minesweeper) - these not only have the potential to alienate women but also international students that may lack the context. Likewise, projects that have latent racism that if not contextualized appropriately, can turn away populations that are targeted in ways that skew the data (i.e., crime rate prediction).

***"The hidden curriculum combined with ambient cues act as warning signals and signs to students, leading to high attrition rates among students whose ideas and voices have been historically absent in computing fields."***

Hidden curriculum and ambient cues are often communicated via our syllabi (policies or absence of policies), assignments (see examples given above), assessments (high stakes assessment, i.e., only a midterm and final), classroom culture (gamification of engagement in the class), assumptions of prior knowledge, and interactions (e.g., instructor-student, student-student). Curricula, both explicit and implicit, are laden with messages to students about who belongs and what can be achieved with computing. To appeal to a broader audience (this includes the invisible identities, e,g, our LGBTQ+ students) a critical examination on both fronts is a necessity.

## Intentionality to Recruit into Post-secondary Education

With exponential growth in computing-related fields, recruitment has not been viewed as a critical matter needing to be addressed. The approach has largely been to manage the numbers of students currently in the field – how to teach more people with the same number of resources. The challenge being that this approach neglects the absence of diversity in the field. The patterns of participation remain largely unchanged with women, Black, Latiné, and Indigenous students grossly underrepresented in the field of computing. Towards the goal of equity, we must explore, develop, and implement intentional means of reaching, inspiring, and recruiting these populations into the field of computing. For example, if your university population is 50% women, what are some strategies that you can leverage to increase engagement with the university population to increase participation of women in computing? Does it include active recruitment in fields that are overrepresented by women e.g., psychology, biology? Does it include the introduction of an X+CS curriculum that bridges these fields to expose more women at the university to this lucrative and impactful field? Likewise, is your institution in proximity of a minority-serving institution (MSI)? Could you be building relationships/curriculum/programs with these institutions that are mutually beneficial that could establish a pathway for their students and expand the diversity of your program?

The key to these initiatives is first understanding the landscape of your institution - local community (who do you serve as an institution?), university (what is the gender, racial, ethnic composition of your university?), departmental (same

questions as university), programmatic (also same questions). This information will allow you to determine low hanging fruit (things you can easily address - e.g., most of our diverse-identifying students come via transfer and we have a requirement that impedes their transition) and more strategic goals (e.g., we have an HBCU in the state that we can and should work on developing a partnership with).

There should also be attention to intersectionality - a department might determine that they have achieved gender parity but upon closer examination their population is still largely White. How can we now work towards diversifying the women in our program (see MSI partnership suggestion above)? This makes the efforts towards diversity explicit and intentional.

# Chapter 5: Beyond the Classroom

We assert that computing has become a fundamental literacy. As such, it is time to recognize that the resources needed to complete a post-secondary education is too high for many people, both in terms of money and time. It is also time to recognize that technological change is rapid, and that continuing education is critical for all citizens to stay abreast of these changes that can adversely impact them. Education outside of the walls of formal education can provide a meaningful way to provide computing-related skills and knowledge, particularly if that education is freely available to all citizens.

## Intentionality to Create Space for and Recruit into Informal Education

There are several types of informal education being currently offered, such as through coding bootcamps, online courses, and even libraries. Some of these are focused on using software or hardware, while others are directly teaching software development and/or programming. To ensure a workforce that is adequately trained in the rapidly changing skill sets needed for computing, it is critical that high-quality resources and supports are put into place to provide this training to any citizen who wants it, whether they are a student studying computing at a postsecondary institution or a person who wishes to change careers after obtaining a postsecondary degree. While providing access to virtual training systems or in-person instruction is a critical first step, it is also vital to encourage citizens to participate in this training.

## Enact Computing as a Fundamental Literacy

diSessa (2000) argued that we need to think about computing in terms of literacy, what he called a "computational literacy" that will have a profound impact on society. Drawing parallels to the societal shift that accompanied the democratization of numerical and algebraic literacy, the emergence of widespread computational literacy can have significant and foundational impacts in how society operates and computing's role in it. In making this argument, diSessa makes clear this vision extends beyond STEM fields, arguing that computational literacy needs to be at the scale of "mass literacy with the written word, which permeates not only all professional intellectual activity in STEM, but almost all learning and instruction in STEM" (diSessa, 2018, p. 4). Similarly, others have argued for computational literacy that should be for everyone and goes beyond technology careers as an endpoint to "include vocational training, civic engagement, and creative expression as possible options" (Kafai & Proctor, 2022, p. 3). Rethinking computational literacy from these perspectives will require moving away from just a focus on computational and technical skills to broadening students' perspectives through a humanizing approach that incorporates social, cultural, and political aspects of technology.

While considering the numerous pathways for computing in postsecondary education and the lifelong education that is needed as the rapidly shiting field of technology, there is a three-fold need to consider educating all citizens about computing as a fundamental literacy. This education is similar to mathematics and language arts, and it must be prioritized

as a matter of equity, justice and national prosperity. First, our democracy relies on the power of its people to make informed decisions in free elections that are secure and safe. These fundamental democratic rights are dependent upon an equitably educated citizenry who are aware of the lack of neutrality in the technology that they use as well as the security and privacy of those technologies.

Second, our nation depends on our technological infrastructure that increasingly contributes to our defense, gross domestic product, labor workforce, economic development, healthcare, financial ecosystem, education, and more. While our fundamental rights in a democratic nation are becoming more and more technologically vulnerable, we are seeing the growth of more people employed by the computing industry. In 2021 the U.S. Department of Labor tracked 10 occupations related to computing, and 9 of these have projected growth of 4% to 35% over the next decade (U.S. Department of Labor, 2022).

Finally, an educated citizenry in which all people have access to, participate in, and have positive experiences learning computing will provide knowledge and economic opportunities to all people as our nation continues to shift from manual to automated labor. A citizenry untrained in computing will be without hope for economic opportunities, and this lack of hope can contribute to civil unrest due to the real as well as perceived injustices.

# Conclusion: Looking Ahead

By recognizing the need for computing to be for all because it impacts all, we can begin to reimagine what the discipline could and should be. We need to make sure that both those within the field of computing and outside understand the impacts of computing in their lives. They will either be the creators or consumers of that technology (or both) and the influence on their lives is not neutral. In addition, more and more jobs will require knowledge of computing, knowledge of how computing looks today, not how computing looked when programming was first introduced in schools. As a community, we need to embark on a journey that will substantially overhaul the way we teach and what we teach. To do so, we should focus on the following:

- Update the curriculum so that it is responsive to the fact that technology will continue to change and that the skills needed by our graduates will change.
- Think critically about what is and is not important in the curriculum. We need to consider what skills and topics enhance the educational experience and what topics are historical remnants.
- Recognize that the above problems can't be solved by simply adding more to the curriculum when new ideas, skills, and topics arise.
- Critically examine what constitutes foundational knowledge about computing and how we can best teach it to students who intend to pursue computing as a major.
- Determine what parts of that foundational knowledge are necessary "for all", recognizing that computing is a fundamental literacy.

- Ensure that all voices are incorporated into and valued in the conversations about technology, particularly when thinking about impacts of technology on society.
- Equip those who are building technology to analyze the impacts of it and assess its risks.

# References and Further Readings


ACM/IEEE-CS. (2013). Computer Science Curricula 2013: Curriculum Guidelines for Undergraduate Degree Programs in Computer Science. ACM, New York.

ACM/IEEE-CS. (2021). Computing Curricula 2020: Paradigms for Global Computing Education. ACM, New York.

Atchison, W. F., Conte, S. D., Hamblen, J. W., Hull, T. E., Keenan, T. A., Kehl, W. B., McCluskey, E. J., Navarro, S. O., Rheinboldt, W. C., Schweppe, E. J., Viavant, W., & Young, D. M. (1981). ACM Recommended Curricula for Computer Science and Information Processing Programs in Colleges and Universities, 1968-1981 [Technical Report]. Association for Computing Machinery.

Benjamin, R. (2019). Race after technology: Abolitionist tools for the new jim code. Social forces.

Braveman, P., & Gruskin, S. (2003). Defining equity in health. Journal of Epidemiology & Community Health, 57(4), 254-258.

Brittain, J.E. (1978). The contemplative EE: Engineering history and education. Proceedings of the IEEE, 66(8), 825–829.

CC2020 Task Force. (2020). Computing Curricula 2020: Paradigms for Global Computing Education. Association for Computing Machinery.

Clements, D. (1999). The Future of Educational Computing Research:The Case of Computer Programming. Information Technology in Child Education. 1. https://www.researchgate.net/publication/255632182_The_Future_of_Educational_Computing_ResearchThe_Case_of_Computer_Programming

Computer Science Curricula 2013 | ACM Other Books. (n.d.). Retrieved November 15, 2023, from https://dl.acm.org/doi/book/10.1145/2534860

Connor, K.A., & Walker, M.F. (1984). The Advent of Electrical Engineering at Rensselaer: 1900-1940. IEEE Transactions on Education, 27(4), 226–231.

Conte, S. D., Hamblen, J. W., Kehl, W. B., Navarro, S. O., Rheinboldt, W. C., Young, D. M., & Atchinson, W. F. (1965). An undergraduate program in computer science - 2014; preliminary recommendations. Communications of the ACM, 8(9), 543–552. https://doi.org/10.1145/365559.366069

Cooper, S., Grover, S., Guzdial, M., and Simon, B. (2014). A future for computing education research. Commun. ACM 57, 11 (November 2014), 34–36. https://doi.org/10.1145/2668899

Denning, P.J. (1981). ACM president's letter: eating our seed corn. Communications of the ACM, 24(6), 341–343.

Denning, P.J., Feigenbaum, E., Gilmore, P., Hearn, A., Ritchie, R.W., and Traub, J. (1981). A discipline in crisis. Communications of the ACM, 24(6), 370–374.

diSessa, A.A. (2018) Computational Literacy and "The Big Picture" Concerning Computers in Mathematics Education, Mathematical Thinking and Learning, 20:1, 3-31.

Dym, B., Pasupuleti, N., Rockwood, C., and Fiesler, C. (2021). "You don't do your hobby as a job": Stereotypes of Computational Labor and their Implications for CS Education. In Proceedings of the 52nd ACM Technical Symposium on Computer Science Education. ACM.

Fiesler, C. (2021). What "counts" as computer science? Blog Mar 18, 2021, accessed Dec 2023. https://cfiesler.medium.com/what-counts-as-computer-science-31f9dd955ad9

Fiesler, C., Friske, M., Garrett, N., Muzny, F., Smith, J.J., and Zietz, J. (2021). “Integrating Ethics into Introductory Programming Classes.” In Proceedings of the 52nd ACM Technical Symposium on Computer Science Education (SIGCSE’21). New York, NY, USA: ACM. 2021.

Frase, K.G., Latanision, R.M., and Pearson, G. (2017). Engineering Technology Education in the United States. National Academies Press.

Goel, V. (2014). Facebook Tinkers With Users’ Emotions in News Feed Experiment, Stirring Outcry. Retrieved from https://www.nytimes.com/2014/06/30/technology/facebook-tinkers-with-users-emotions-in-news-feed-experiment-stirring-outcry.html

Goldweber M., Little J., Cross G., Davoli R., Riedesel C., von Konsky B., and Walker H. (2010). Enhancing the Social Issues Components in our Computing Curriculum. Proceedings of the 2010 ITiCSE working group reports. (117-133). https://dl.acm.org/doi/10.1145/1971681.1988996

Granger, M. J., Little, J. C., Adams, E. S., Björkman, C., Gotterbarn, D., Juettner, D. D., ... and Young, F. H. (1997). Using information technology to integrate social and ethical issues into the computer science and information systems curriculum (report of the ITiCSE'97 working group on social and ethical issue in computing curricula). In The supplemental proceedings of the conference on Integrating technology into computer science education: working group reports and supplemental proceedings. New York, NY, USA: ACM. 1997, pgs. 38-50.

Grayson, L.P. (1978). Engineering education throughout the world: A synoptic view. Proceedings of the IEEE, 66(8), 940–956.

Grayson, L.P. (1980). A Brief History of Engineering Education in the United States. IEEE Transactions on Aerospace and Electronic Systems, AES-16(3), 373–392.

Hammonds, E., Taylor, V., and Hutton, R. (2021). Transforming Trajectories for Women of Color in Tech. National Academies Press.

Jørgensen, U. (2007). Historical Accounts Of Engineering Education. In: Rethinking Engineering Education. Springer, Boston, MA. https://doi.org/10.1007/978-0-387-38290-6_10

K–12 Computer Science Framework. (2016). http://www.k12cs.org

Kafai, Y. B., & Proctor, C. (2021). A Revaluation of Computational Thinking in K–12 Education: Moving Toward Computational Literacies. Educational Researcher, 0013189X2110579.

Kaspar, J., Harrendorf, S., Butz, F., Höffler, K., Sommerer, L., Christoph, S. (2023). Artificial Intelligence and Sentencing from a Human Rights Perspective. In: Završnik, A., Simončič, K. (eds) Artificial Intelligence, Social Harms and Human Rights. Critical Criminological Perspectives. Palgrave Macmillan, Cham. https://doi.org/10.1007/978-3-031-19149-7_1

Ko, A. J., Oleson, A., Ryan, N., Register, Y., Xie, B., Tari, M., David- son, M., Druga, S., & Loksa, D. (2020). It is time for more critical CS education. Communications of the ACM,

Lucia, O., Martins, J., Ibrahim, Y., Umetani, K., Gomes, L., Hiraki, E., Zeroug, H., and Manic< M. (2021). Industrial Electronics Education: Past, Present, and Future Perspectives. IEEE Industrial Electronics Magazine, 15(1), 140–154.

Margolis, J., & Fisher, A. (2003). Unlocking the clubhouse: Women in computing. The MIT Press.

Margolis, J. (2008). Stuck in the shallow end: Education, race, and computing. The MIT Press.

Martin, C.D. (1997). The case for integrating ethical and social impact into the computer science curriculum. In The supplemental proceedings of the conference on Integrating technology into computer science education: working group reports and supplemental proceedings (ITiCSE-WGR '97). Association for Computing Machinery, New York, NY, USA, 114–120. https://doi.org/10.1145/266057.266131

McNamara, A., Smith, J., & Murphy-Hill, E. (2018). Does ACM's code of ethics change ethical decision making in software development? In Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE 2018). Association for Computing Machinery, New York, NY, USA, 729–733. https://doi.org/10.1145/3236024.3264833

Miaskiewicz, T., & Kozar, K. A. (2011). Personas and user-centered design: How can personas benefit product design processes? Design Studies, 32(5), 417–430.

Mowery, D.C., and Langlois, R.N. (1996). Spinning off and spinning on(?): the federal government role in the development of the US computer software industry. Research Policy, 25(6), 947-966.

Nathan, M. J., & Petrosino, A. (2003). Expert Blind Spot Among Preservice Teachers. American Educational Research Journal, 40(4), 905–928.

National Science Foundation. (2018). National Science Foundation Strategic Plan for 2018-2022: Building the Future. Retrieved December 2023 from https://www.nsf.gov/pubs/2018/nsf18045/nsf18045.pdf

Núñez, A., Mayhew, M. J., Shaheen, M., & Dahl, L. S. (2021). Let's teach computer science majors to be good citizens. The whole world depends on it. Edsurge.

O'Neil, C. (2016). Weapons of math destruction: How big data increases inequality and threatens democracy. Broadway Books.

Postman, N. (1998). Five Things We Need to Know About Technological Change. Retrieved December 2023 from https://web.cs.ucdavis.edu/~rogaway/classes/188/materials/postman.pdf

Seely, B. E. "SHOT, the History of Technology, and Engineering Education." Technology and Culture, vol. 36, no. 4, 1995, pp. 739–72. JSTOR, https://doi.org/10.2307/3106914.

Singer, N., and Huang, K. (2022). Computer Science Students Face a Shrinking Big Tech Job Market. Retrieved December 2023 from https://www.nytimes.com/2022/12/06/technology/computer-students-tech-jobs-layoffs.html

Singer and Natalie R. Nielsen and Heidi A. Schweingruber (2012). Discipline-Based Education Research. National Academies Press.

Steier, R. (1983). From Washington: Replanting the "Seed Corn". Communications of the ACM, 26(4), 245.

Tissenbaum, M., Weintrop, D., Holbert, N., & Clegg, T. (2021). The Case for Alternative Endpoints in Computing Education. British Journal of Educational Technology, 52(3), 1164–1177.

U.S. Department of Labor. (2022). Occupational Outlook Handbook: Computer and Information Technology Occupations. Retrieved December 2023 from https://www.bls.gov/ooh/computer-and-information-technology/home.htm

Watters, A. (2021). Teaching machines: The history of personalized learning. MIT Press.

Yadav, A., & Heath, M. K. (2022). Breaking the Code: Confronting Racism in Computer Science through Community, Criticality, and Citizenship. TechTrends, 66(3), 450–458.

Zweben, S. and Bizot, B. 2020 Taulbee Survey: Bachelor's and Doctoral Degree Production Growth Continues but New Student Enrollment Shows Declines. Computing Research News, May 2021: 2–68