The **Promise** and **Problems** of CS for All

Amy J. Ko, Ph.D.
The Information School
University of Washington, Seattle

Computing is **everywhere** and **everything**, for better and worse.

It’s transforming health and medicine

It enables globalized private surveillance infrastructure

It enables creative expression.

It isolates us.


It empowers


It disempowers
NEXT ACADEMY
eat
sleep
code

Everyone should learn about this fierce and fraught medium.

Amy J. Ko, Alannah Oleson, Mara Kirdani-Ryan, Yim Register, Benjamin Xie, Mina Tari, Matt Davidson, Stefania Druga, Dastyni Loksa, Greg Nelson (2020). It's Time for More Critical CS Education. Communications of the ACM (CACM), 31-33. https://doi.org/10.1145/3424000

In higher education, **many** do.

- At some colleges and universities, **1/3rd** of students major in CS (!) — almost 20% at MIT
- Most CS departments are **overwhelmed** with demand
- Demand has led to **secondary markets** such as bootcamps, corporate training, online degrees, etc.
- Scale, and a commitment to “merit” has also exacerbated deeply rooted problems with diversity, equity, and inclusion.

Computing Research Association (2017). Generation CS: Computer Science Undergraduate Enrollments Surge Since 2006. http://cra.org/data/Generation-CS/

But in K-12, **few** students learn computing.

- Across the U.S., our best data shows that **<30%** of schools offer CS electives
- … and **<1%** of students take a class.
- And most in North America who do are wealthy **white**, **Chinese**, and **Indian boys**, many of whom have family or friends in computing, or whose parents expect them to pursue tech to support their families.

Code.org (2021). 2021 Annual Report. https://code.org/files/code.org-annual-report-2021.pdf

Why such disparities between **higher ed** and **K-12**?

Margolis, Jane. Stuck in the Shallow End: Education, Race, and Computing. MIT press, 2017.
https://dl.acm.org/doi/book/10.5555/3153292

It's partly **structural**.

- Unequal paths to develop **interest**.
- Unequal **capacity** for CS in schools.
- Unequal pathways to **college**.
- Unequal access to the **internet**.

But it's also **pedagogical**.

Despite teaching CS for decades, we don't know how to equitably and effectively **teach**, prepare **teachers**, make students feel **welcome,** make CS **relevant** to everyone, **assess** knowledge, **scale** learning, …

And thus the **status quo**…

- CS education tends to filter out diversity through narrow notions of **rigor** and **merit**
- CS education concentrates **power** and **wealth** amongst white and Asian men
- The public lacks **basic literacy** about CS and how it concentrates power and wealth
- We lack sufficient **research** to inform change
- We lack sufficient **capacity** to implement change

But there is **hope**!

- **15+ years** of research funding for basic and applied research in the US, UK, EU, Japan, Korea, China...
- A global community of researchers, teachers, and activists that has grown an **order of magnitude** in the past decade.
- A public that is realizing the importance of **CS literacy** and beginning to wonder why youth (and politicians… even engineers) aren't learning it.

This **talk**

Why **I** started doing computing education, after 15 years in HCI.

What I've **discovered** about structural and pedagogical issues in teaching CS.

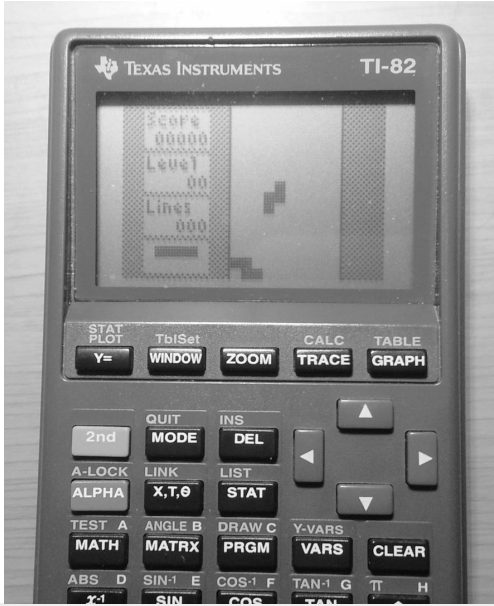
1. *What is CS knowledge?*
2. *How should we teach it?*
3. *How do we include everyone?*

What **grand challenges** remain in research and practice.

My unexpected path to **computing education research**.

I learned to code because of **pre-algebra**.

- My math teacher required us to have a **TI-82 graphing calculator**.
- A classmate showed me a version of **Tetris** his older brother had acquired. But it was too slow!
- I spent a summer reading the manual, and rewriting the **renderer**, so I could play in class.
- I shared with my classmates, became their **hero**, was praised by my teacher, and fell in love with computing's capacity for creative expression.



I studied **CS** + **Psychology** in the 90's

- CS because I was **poor** and needed to make money
  - Most of what I learned was incredibly boring.
  - Classes leeched all of the joy from programming.
  - Most of my professors were unskilled teachers.
  - I watched many peers drop out out of boredom, confusion
- Psychology because **behavior** was fascinating
  - I was captivated in every class.
  - It explained so much of the world.
  - But I couldn't get paid to study it.

Or could I?

- I discovered research!
- I learned I could study **programming** for $.
- I blended **human-computer interaction** and **software engineering**, studying struggles to understand code and inventing ways to make it easier.
- I earned my PhD at Carnegie Mellon, **inventing**, **theorizing**, **observing**, and **writing** about programming, then continued as a professor.

After tenure, I co-founded a **startup**.

I learned two things as CTO managing 8 engineers:

- Understanding code **is** hard.
- But it's hard because **learning** is hard.

Nearly every difficulty my engineers faced was because they **struggled to learn** a new programming language, API, platform, or how to collaborate. When I found ways of teaching them well, they excelled.

So in ~2012, I **pivoted** to computing education

I found a growing, passionate, collaborative community of **computing education researchers** who also wondered:

- Why is learning to code so hard?
- Why is CS mostly white and Asian boys?
- Why do so many students drop out of CS?
- How can we teach CS more equitably and inclusively?
- How can tools help with teaching + learning?

Here's what my lab and I have **discovered** in the past decade.

What is **CS knowledge**?

It’s **not** what you think.

We in higher education usually think of CS as:

- Programming languages
- Data structures
- Algorithms
- Theory of computation
- Artificial intelligence
- Systems, etc.

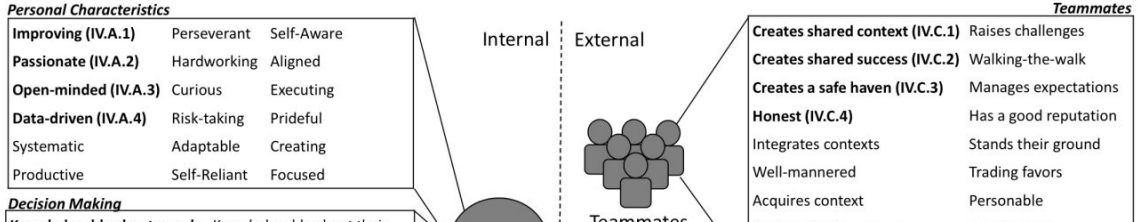
It is technical, but it is also **cognitive**, **social**, and **political**.

Paul Luo Li, et al. (2019). What Distinguishes Great Software Engineers? Empirical Software Engineering.

Paul Luo Li et al. (2017). Cross-Disciplinary Perspectives on Collaborations with Software Engineers. IEEE CHASE.

Paul Luo Li, et al. (2015). What Makes a Great Software Engineer? ACM/IEEE ICSE.

Paul Li interviewed + surveyed **2,000**+ software engineers, and while they viewed CS knowledge as core, they often viewed it as less important than the ability to make **complex technical decisions** in the context of organizational, market, and political **uncertainties**.

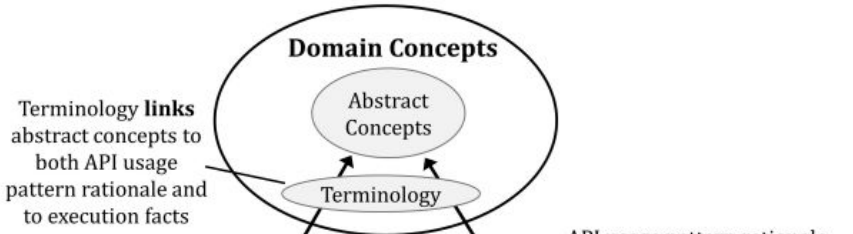


In practice, CS is also more about **API learning** and than algorithm design.

Kyle Thayer et al. (2021). A Theory of Robust API Knowledge. ACM TOCE.

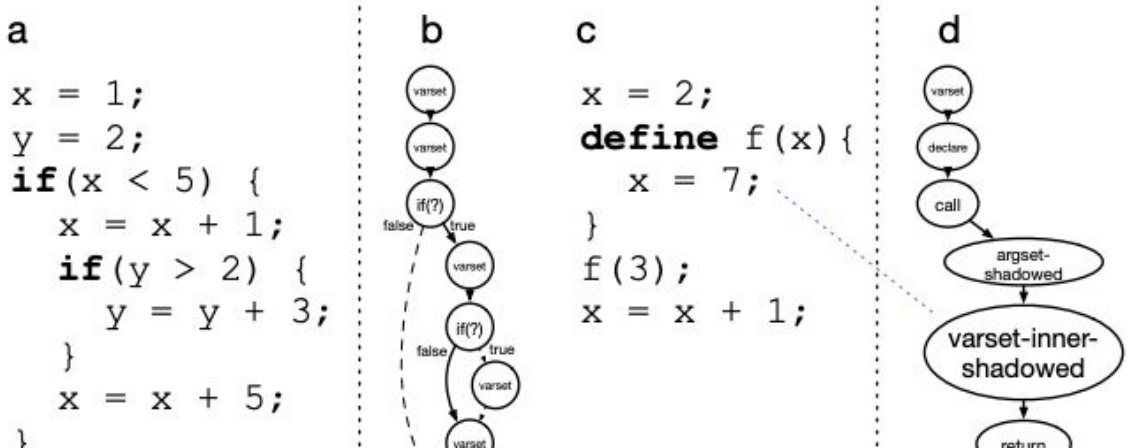
Kyle Thayer et al. (2020) Barriers Faced by Coding Bootcamp Students. ACM ICER.

Kyle Thayer studied students in coding bootcamps and found that **API learning** dominated their time, far more than programming language learning. He developed a theory demonstrating that API knowledge quite unlike other kinds of learning, and often not well supported in or out of school



And despite our best efforts in CS to teach programming languages, we often **fail**.

Greg Nelson found that students' **programming language semantics** knowledge is often far more brittle than we think, and predicts and explains much of later failure in CS education.



Greg Nelson et al. (2020) Towards Validity for a Formative Assessment for Language-Specific Program Tracing Skills. ACM Koli Calling.

Benjamin Xie et al. (2019). A Theory of Instruction for Introductory Programming Skills. Computer Science Education.

And in K-12, CS is being embraced framed more **broadly**.

Alannah Oleson et al. (2020). On the Role of Design in K-12 Computing Education. ACM TOCE.

Alannah Oleson analyzed CS learning standards and curricula and found that schools, teachers, and instructional designers lean *hard* on **design skills** because creativity resonates more with students than algorithms and data structures. But they call design skills “CS”.

Problem-Space Design					Program-Space Design				
UtC	*CI*	*ES*	*II*	*Comm*	*UtC*	*CI*	*ES*	*II*	*Comm*
Code.org CSD									
✓	✓	✓	✓	✓	✓	✓			
✓	✓				✓	✓	✓	✓	✓
✓	✓	✓		✓	✓	✓	✓	✓	✓
✓	✓	✓	✓	✓	✓	✓		✓	✓
✓	✓	✓	✓	✓		✓		✓	
✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
AP CS A									
							✓	✓	
						✓	✓		

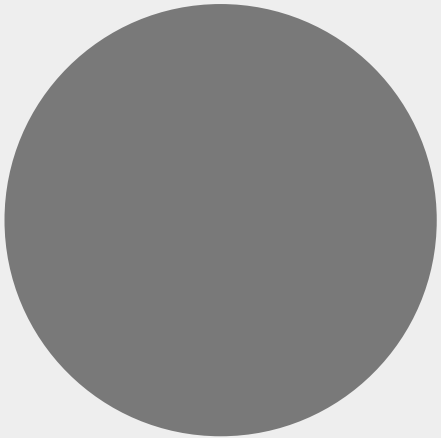
CS broadly excludes **social**, **ethical**, **political**, and **justice** aspects of computing.

Amy J. Ko , et al. (2020). It's Time for More Critical CS Education. CACM.

The past two decades of social science has revealed many **structural forms of bias and inequity**, some *amplified* by computing, some *created* by it.

But none of this is taught at any level of education. Calls to teach it have only just emerged, first by social scientists, then education researchers. My lab has brought that call to CS more broadly.

So what is **CS knowledge**?

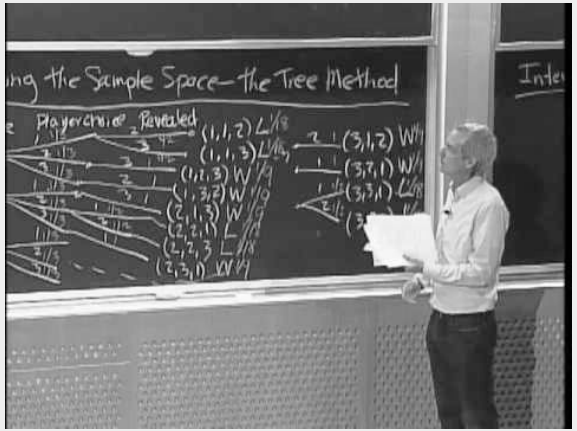


The usual topics, **but also**…

- **Language semantics** we rarely teach well, but aggressively assess.
- **Problem solving skills** we rarely teach at all, but aggressively assess.
- **API learning skills** we assume are trivial.
- **Design skills** that resonate deeply with youth, but that deemphasize.
- **Decision making skills** we rarely teach *or* assess, but are crucial in industry.
- **Diversity literacy** that is essentially ignored, perpetuating oppression.

How **should** we teach CS?

The typical pedagogy in CS classes involves…



- Teacher **explains** concepts, expects transfer to programming skills.
- Transfer does not happen, so students learn skills **independently**, with each other, online, and/or in office hours.
- Students are often punished for this behavior under the guise of **academic misconduct**.
- The only students who survive this process are ones who arrive with **prior knowledge**.

We've known this doesn't work for **decades**. So what does?

Greg Nelson et al. (2019). Towards Validity for a Formative Assessment for Language-Specific Program Tracing Skills. ACM Koli Calling.

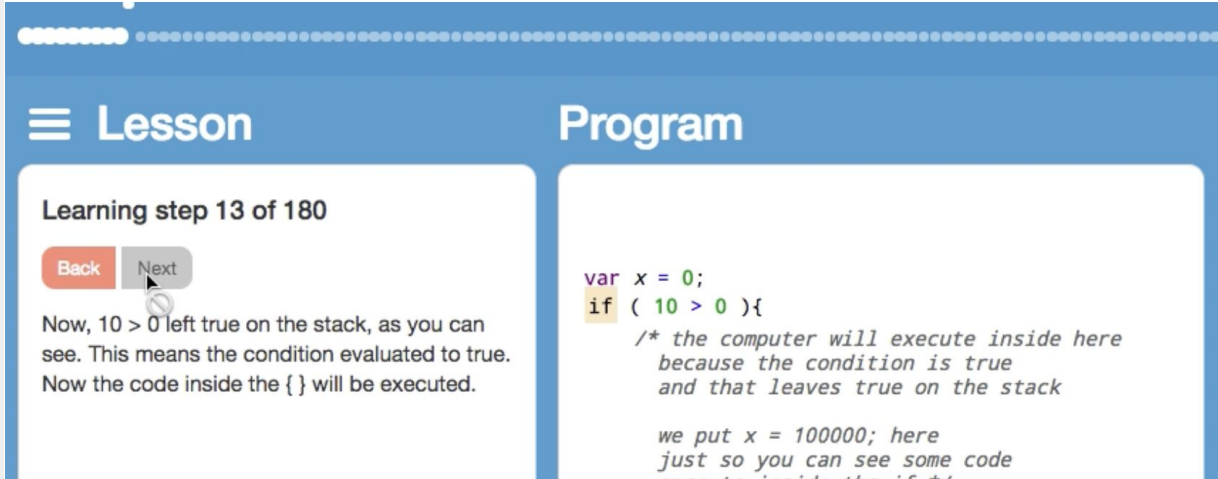
Benjamin Xie et al. (2019). A Theory of Instruction for Introductory Programming Skills. Computer Science Education.

Michael J. Lee, et al. (2015). Comparing the Effectiveness of Online Learning Approaches on CS1 Learning Outcomes. ACM ICER.

Michael J. Lee, et al. (2014). Principles of a Debugging-First Puzzle Game for Computing Education. IEEE VL/HCC.



Mike Lee, Benji Xie, and Greg Nelson found that teaching program **reading before writing**, and **explicitly assessing reading** skills, can be effective at promoting robust *writing* skills.



Teaching **explicit programming strategies** can help too.

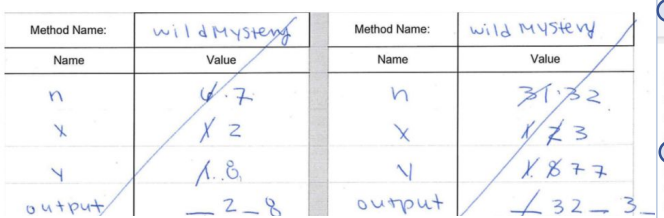
Thomas D. LaToza et al. (2020). Explicit Programming Strategies. Empirical Software Engineering.

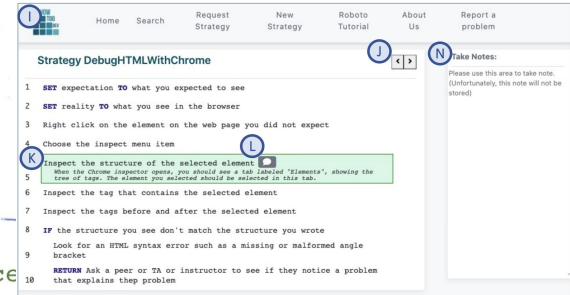
Maryam Arab et al. (2021). HowToo: A Platform for Sharing, Finding, and Using Programming Strategies. IEEE VL/HCC.

Maryam Arab et al. (2022). An Exploratory Study of Sharing Strategic Programming Knowledge. ACM CHI.

Benjamin Xie et al. (2018). An Explicit Strategy to Scaffold Novice Program Tracing. SIGCSE.

Thomas Lazota and Maryam Arab, we've found that scaffolding problem solving with **step-by-step procedures** can help novices match the performance of experts.






```
1  STRATEGY renameVariable (name)
2    SET codeLines TO all lines of source
3    FOR EACH 'line' IN codeLines
4      IF the line contains a valid reference to the variable
5        Rename the reference
6    SET docLines TO all lines of documentation that contain the name 'name'
7    FOR EACH 'line' IN docLines
8      IF the line contains a reference to the name
```

Prosocial feedback is key to self-efficacy.

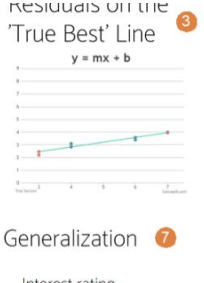
Yim Register et al. (2020). Learning Machine Learning with Personal Data Helps Stakeholders Ground Advocacy Arguments in Model Mechanics. ACM ICER.

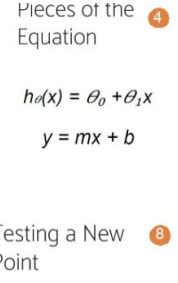
Michael J. Lee et al. (2013). In-Game Assessments Increase Novice Programmers' Engagement and Learning Efficiency. ACM ICER.

Michael J. Lee et al. (2012). Investigating the Role of Purposeful Goals on Novices' Engagement in a Programming Game. IEEE VL/HCC.

Michael J. Lee et al. (2011). Personifying Programming Tool Feedback Improves Novice Programmers' Learning. ACM ICER.

Mike Lee and Yim Register found that subtle changes in feedback — using **personal pronouns**, redirecting **blame** to the machine, using **personal data**, even giving compilers **eyes** — causes students to attend more carefully to instruction, improving learning.



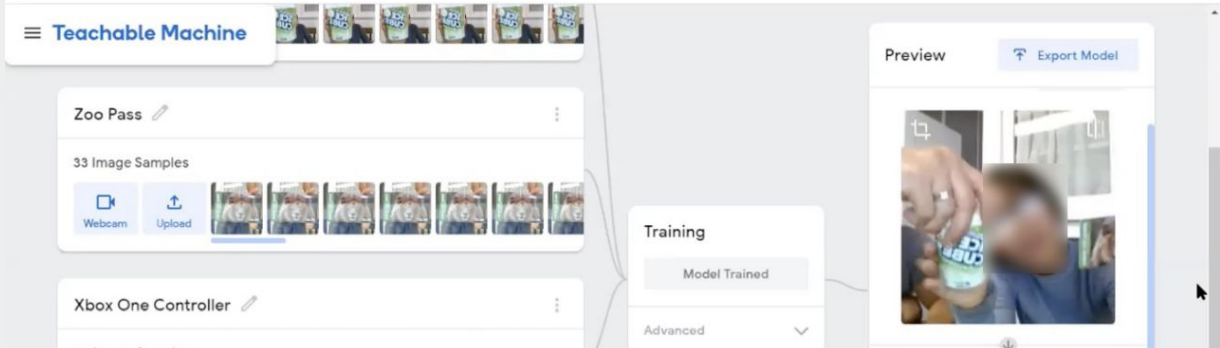


Engaging youth in creating with AI, especially in with family, quickly dispels **AI hype**.

Stefania Druga, Amy J. Ko (2021). How Do Children's Perceptions of Machine Intelligence Change when Training & Coding Smart Programs? ACM IDC.

Stefania Druga, Fee Christoph, Amy J. Ko (2022). Family as a Third Space for AI Literacies: How Do Children and Parents Learn about AI Together? ACM CHI.

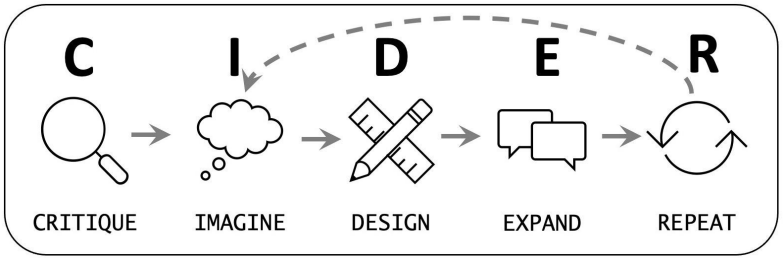
Stefania Druga has found that when youth collaborate with family to train classifiers, they quickly come to see how brittle AI be, and how responsible its creators are for deciding who it does and doesn't serve.



Teaching design skills can benefit greatly from focusing on **assumptions**.

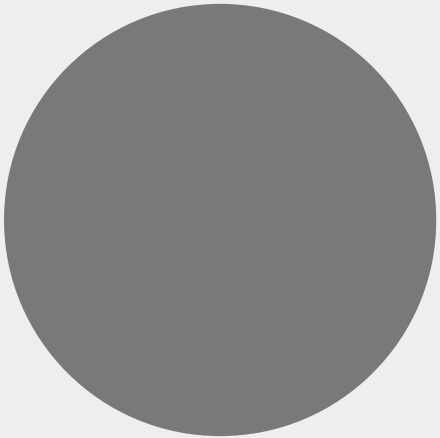
Alannah Oleson, Meron Solomon, Christopher Perdriau, Amy J. Ko (2022). Teaching inclusive design skills with the CIDER assumption elicitation technique. ACM ToCHI.

Alannah Oleson designed the **CIDER** teaching method, which systematically develops students' ability to identify assumptions made in a software design by showing them assumptions that they *didn't* notice that other students *did*.



Want to try it in your class? Sign up for AI's study!

So how **should** we teach CS?



Quite differently than we do now:

- Use **active learning**, with targeted, personalized, in situ direct instruction on skills
- More **formative feedback** to diagnose what students do and don't know; less summative.
- More **explicit scaffolding** of programming skills, less "figure it out yourself, alone."
- Centering **design** and **diversity** in how we define and contextualize CS foundations.

How do we **include** everyone?

It’s more than just adding outreach programs, and tweaking curricula. It requires reconsidering **foundations**.

CS has notions of **rigor** (merit) and **epistemology** (positivism) that *cause* exclusion.

Some in CS also have **political opposition to notions of equity** , viewing any effort to ensure students have what they need to learn as “coddling” or “lowering standards”.

Here are a few examples of these deep cultural tensions…

Peer mentorship is fundamental to developing belonging and identity in CS

Amy J. Ko et al. (2018). Informal Mentoring of Adolescents about Computing: Relationships, Roles, Qualities, and Impact. ACM SIGCSE.

Amy J. Ko et al. (2017). Computing Mentorship in a Software Boomtown: Relationships to Adolescent Interest and Beliefs. ACM ICER.

Harrison Kwik et al. (2018). Experiences of Computer Science Transfer Students. ACM

With many undergrads, I have shown that **peer relationships** are essential. Students report that strict rules against collaboration, designed to "accurately measure merit", disrupt their ability to form community by creating a culture of **competition** and **peer comparison**.

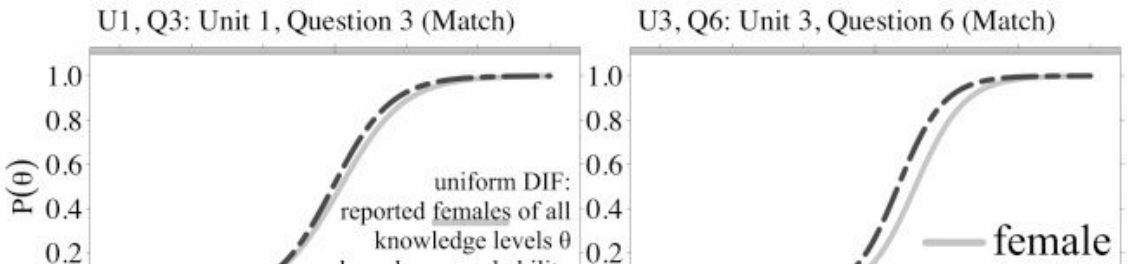
Assessments in CS are often biased in ways difficult to see without **psychometrics** expertise.

Benjamin Xie et al. (2021). Domain Experts' Interpretations of Assessment Bias in a Scaled, Online Computer Science Curriculum. ACM Learning at Scale.

Benjamin Xie et al. (2019). An Item Response Theory Evaluation of a Language-Independent CS1 Knowledge Assessment. ACM SIGCSE.

Matt Davidson et al. (2021). Investigating Item Bias in a CS1 exam with Differential Item Functioning. ACM SIGCSE.

Benji Xie and Matt Davidson have shown how tests used in CS classes are viewed as objective, but actually have **systematic racial and gender biases** that impose structural disadvantages to students with marginalized identities.

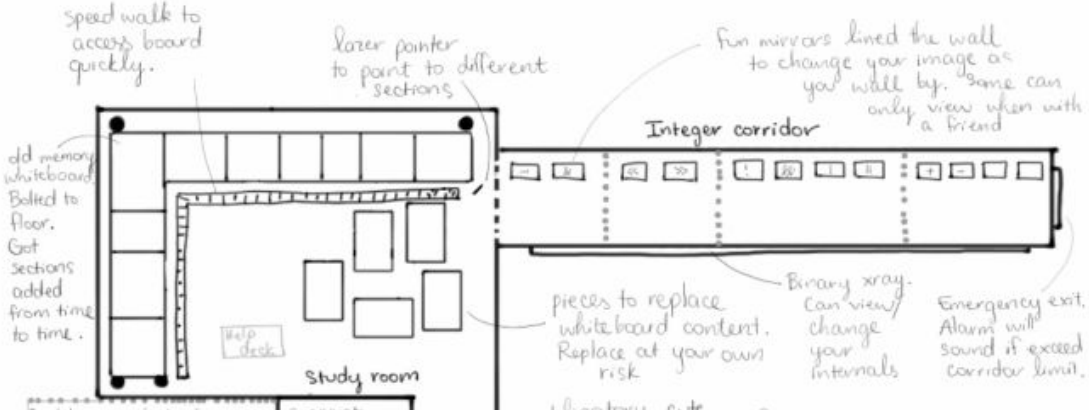


Integrating **social**, **ethical**, and **political** topics can engage marginalized students.

Mara Kirdani-Ryan et al. (2022). The House of Computing: Integrating Counternarratives into Computer Systems Education. ACM SIGCSE.

Mara Kirdani-Ryan et al. (in review). "Taught to be automata": Examining the departmental

But Mara Kirdani-Ryan has found that students with dominant identities are often **resistant** to such learning, deeming it off topic, irrelevant to jobs. But these sentiments come from faculty.



Talking about CS in social, political, and ethical terms requires a sense of **safety**.

Jayne Everson et al. (2022). "A key to reducing inequities in like, AI, is by reducing inequities everywhere first": Emerging Critical Consciousness in a Co-Constructed Secondary CS Classroom. ACM SIGCSE.

Jayne Everson and Megumi Kivuva found in one study that a class of adolescents marginalized in CS **didn't feel safe** talking about CS critically until they were confident that teachers respected their lived experiences and shared their values about schools, CS, and technology.

Prospective CS teachers **internalize fears** about CS, rigor, and failure.

Jayne Everson et al. (2022). "I would be afraid to be a bad CS teacher": Factors Influencing Participation in Pre-Service Secondary CS Teacher Education. ACM ICER.

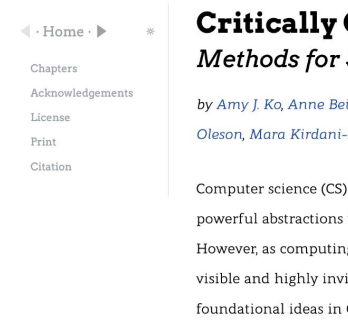
Jayne also found that a key barrier to aspiring teachers wanting to teach CS is a sense that they would **not belong**, they would be **judged**, and worse yet, they would end up **perpetuating** the same negative learning cultures they had experienced in CS in college.

We also have to explore *how* to teach CS in **sociopolitical** ways.

Amy J. Ko , Anne Beitlers, Brett Wortzman , Matt Davidson , Alannah Oleson , Mara Kirdani-Ryan , Stefania Druga , Jayne Everson (2021). Critically Conscious Computing: Methods for Secondary Education . https://criticallyconsciouscomputing.org

We wrote a book, **Critically Conscious Computing: Methods for Secondary Education**, to support our teacher education efforts.

Across 25 chapters, oit reframes CS in technical *and* sociopolitical terms (e.g, *how if-statements perpetuate poverty*)



None of this happens without **excellent teachers**.

Amy J. Ko et al. (2023, **to appear**). Proposing, Preparing, and Teaching an Equity- and Justice-Centered Secondary Pre-Service CS Teacher Education Program. SIGCSE.

Anne Beitlers and I launched **STEP CS** last Spring, a teacher certification program that prepares equity and justice-centered secondary CS educators:

https://computinged.uw.edu/stepcs/

Teachers — including those of us in higher education — also need ongoing **professional development**.



Alannah, Richard Ladner, and I are co-editing a new book, **Teaching Accessible Computing**, which teaches CS faculty how to integrate accessibility topics into *all* areas of CS teaching.

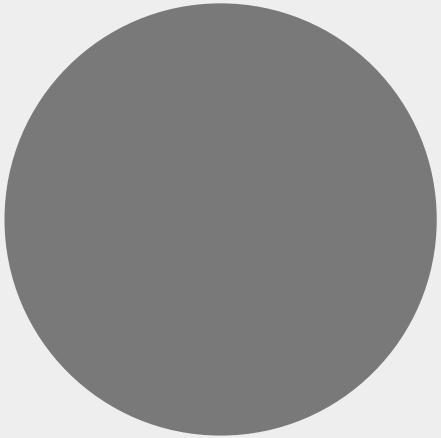
- Do you want to help write it? Email me.
- Do want to read it? We hope to release in **Autumn 2023**.

… and none of this happens without new **languages and tools**, as current ones exclude people with disabilities or who aren't English fluent.

On sabbatical, I am designing a new language, editor, and platform that includes **all abilities and all natural languages**. This requires a new language, editor, debugger, documentation, etc., as current ones often ignore accessibility, Unicode, and the ways that these interact with culture, ethnicity, and expression.

```
↓ ⌨
↓ words
`The game state is a list of guesses and a secret word.`/eng
`El estado del juego es una lista de conjeturas y una palabra secreta.`/spa
•Game(guesses•[""] secret•"")
(
  ƒ guessesRemaining() (secret.length() · 2) - guesses.length()
  ƒ status()
    secret = "" ? "start"
    guessesRemaining() ≤ 0 ? "lost"
    secret→[""].all(f(letter•"") guesses.has(letter)) ? "won"
```

So how do we **include** everyone?



Fundamentally, it means:

- Replacing notions of **rigor** and **merit** in CS with more pluralist epistemologies
- Abandoning anti-collaborative assessment practices, which are **systematically biased** against marginalized students
- Centering **identity**, **equity**, **inclusion**, and **politics** in teaching
- Creating CS teacher education **pathways** and **opportunities**
- Building more inclusive **programming language stacks**

What's **next**?

These are just the things **my lab** has learned.

There are **hundreds** of computing education research papers published every year that deepen our knowledge of problems in CS and ways to address them.

Some of these discoveries are **reshaping** how we think about what and who computing education is for…

… so what is CS **for**, if not supporting industry?

- Ensuring our future politicians, doctors, and HR managers know that AI isn't **infallible magic**.
- Educating a public that knows **when and when not** to use data and algorithms to solve problems
- Educating engineers that have a deep humility about their **ignorance** about how everyone else lives and what everyone else values.

These visions raise questions about **school**

- What kind of **literacies** about computing are needed and possible for a functioning democracy?
- How do we prepare not only more CS teachers, but **excellent, equity and justice-focused** CS teachers, at all levels?
- What knowledge do educators need to bring **racial, gender, and ability justice** to their computing classrooms?

These visions raise questions about **capitalism**

- Who does **industry** involvement in the CS curricula benefit and what other ways might we resource and shape school?
- What role might **automation** play in all of this, if any? Or is automation inherently problematic in learning?
- What **incentive** does industry have to support any equity goals in CS education, other than superficially bolstering their reputation?

Are you CS faculty?

- **Join us!** It took me years to gain competence in education + learning sciences, but a pivot is possible and fun. There's lots of $, wonderful students, and endless challenging, open research questions.
- But come with **humility**. There are a hundred years of scholarship about teaching, learning, and education, and most CS faculty know little of it (and often believe long disproved myths about about learning).

Are you education faculty?

- Although CS is not yet compulsory in schools, it is **less ignorable** every day. Now is the time to shift some of our precious attention — and money — to promoting computing literacy.
- **Hire** tenure-track CS education faculty, **integrate** CS into teacher education programs, and **grow** a robust community of scholars. The University of Washington, Seattle is doing it, why aren't you? 😉

Are you a CS student?

We need contributions at all levels:

- **Teachers** and school leaders at all levels
- Instructional and curriculum **designers**
- **Designers** and **engineers** of CS ed tech
- Policy **experts**
- Computing education **researchers**

Pathways for all of these careers are emerging now.

Thank you!

Summary

- CS isn’t what you think it is.
- If you teach CS, you probably are doing it poorly without knowing it.
- Including everyone means reinventing CS, rigor, merit, progress, purpose, and tools.
- Come join us! We throw good parties :)