

University of Nebraska - Lincoln

DigitalCommons@University of Nebraska - Lincoln

CSE Conference and Workshop Papers

Computer Science and Engineering, Department
of

11-15-2022

Computer Engineering Education

Marilyn Wolf

University of Nebraska-Lincoln, mwolf@unl.edu

Follow this and additional works at: <https://digitalcommons.unl.edu/cseconfwork>



Part of the [Computer Engineering Commons](#), [Electrical and Computer Engineering Commons](#), and the [Other Computer Sciences Commons](#)

Wolf, Marilyn, "Computer Engineering Education" (2022). *CSE Conference and Workshop Papers*. 338.
<https://digitalcommons.unl.edu/cseconfwork/338>

This Article is brought to you for free and open access by the Computer Science and Engineering, Department of at DigitalCommons@University of Nebraska - Lincoln. It has been accepted for inclusion in CSE Conference and Workshop Papers by an authorized administrator of DigitalCommons@University of Nebraska - Lincoln.



Computer Engineering Education

Marilyn Wolf , University of Nebraska-Lincoln

Computer engineering is a rapidly evolving discipline. How should we teach it to our students?

This virtual roundtable on computer engineering education was conducted in summer 2022 over a combination of email and virtual meetings. The panel considered what topics are of importance to the computer engineering curriculum, what distinguishes computer engineering from related disciplines, and how computer engineering concepts should be taught.

COMPUTER ENGINEERING

COMPUTER: Welcome to this virtual roundtable on computer engineering education. A good place to start seems to be to define computer engineering. What is a concise definition of the field that captures where it is and where the field is going?

Digital Object Identifier 10.1109/MC.2022.3205936
Date of current version: 15 November 2022

GREG BYRD: Computer engineering equals the design and analysis of computing hardware and software, both individually and as components in a system.

ROBERT DICK: Yeah. It necessarily spans algorithms and physical implementation substrates. As for

another possible definition, just the facts: design, analysis, and implementation of computer systems. Where it's going: creating easy-to-use computer systems that help people by automating mundane tasks, organizing and sharing information, connecting them with the physical world, and magnifying their intellectual and physical abilities.

JAY BROCKMAN: Broadly, I think that computer engineering is the field of designing machines that process symbolic information. This has fairly vague boundaries that overlap with other established fields, in particular, computer science and electrical engineering, but other fields as well. At this point in time, I think the field is still centered upon the design of a specific kind of machine, namely, digital computers that operate on symbolic information encoded as 1s and 0s. The term *designing* includes coming up with the organization of computing systems themselves as well as the development of tools

ROUNDTABLE PANELISTS

John M. Acken is a faculty member in the electrical and computer engineering (ECE) department at Portland State University. Acken received a B.S. and an M.S. in electrical engineering from Oklahoma State University and a Ph.D. in electrical engineering from Stanford University. He is a Member of IEEE, Eta Kappa Nu, and Tau Beta Pi.

Jay Brockman is professor of the practice and director of the Center for Civic Innovation at the University of Notre Dame. He cofounded Lucata, Inc. and wrote an introductory engineering textbook. Brockman received a Ph.D. from Carnegie Mellon University.

Greg Byrd is professor and associate department head of ECE at North Carolina State University. Byrd received a Ph.D. in electrical engineering from Stanford University. He teaches undergraduate programming, and his research interests include quantum computing and parallel systems.

Robert Dick is an associate professor of electrical engineering and computer science at the University of Michigan. He cofounded Stryd, a wearable electronics company, and was a visiting professor at Tsinghua University. His research focuses

on embedded systems. Dick received a Ph.D. from Princeton University and a B.S. from Clarkson University. He is a Senior Member of IEEE.

David Harris is the Harvey S. Mudd Professor of Engineering Design at Harvey Mudd College. Harris received a Ph.D. in electrical engineering from Stanford University. His professional interests include integrated circuits, microprocessors, computer arithmetic, hiking guidebooks, and experimental aviation.

Jan Madsen is a professor in computer-based systems and interim director of the Department of Applied Mathematics and Computer Science at the Technical University of Denmark (DTU). Madsen received an M.Sc. in electrical engineering and a Ph.D. in computer science from DTU. He is a Member of IEEE, the Association for Computing Machinery (ACM), and the board of the European Design and Automation Association.

Mani Srivastava is a professor of ECE at the University of California, Los Angeles. Srivastava received a Ph.D. from the University of California, Berkeley. He is a Fellow of IEEE and ACM.

and methodologies used in designing computer systems. I personally feel that this is the core that computer engineering shouldn't lose sight of as other considerations come into play and as technologies evolve.

JAN MADSEN: I agree that it is important to have a broader view of the definition of computing machines. Two important emerging fields are reshaping the substrates in which we can build machines: quantum computing and synthetic biology. I think it is important to start introducing these topics in our CE (computer engineering) curriculum.

DAVID HARRIS: Another definition of computer engineering is the *design*

and implementation of digital systems to meet societal needs. Breaking that down, I propose that engineers are people who produce systems that meet societal needs. Meeting societal needs distinguishes engineering from science or other fields primarily concerned about advancing knowledge. Analysis is a means to that end, rather than an end in itself. Those primarily focused on analysis might be wearing mathematician or computer scientist hats. I like Jay's definition too, and especially the part about the vague and overlapping boundaries. Nevertheless, I think we should look for a definition that distinguishes computer engineering from computer science. But the fields overlap enough that you can't look at

a person's actions and classify them unambiguously as CE or CS (computer science). I've wrestled with "digital" versus "symbolic." I agree with Jay that they are largely synonymous in contemporary practice and tend to feel digital is clearer to a nonspecialist audience. I've also wrestled with "digital" versus "computer." I'd prefer a definition of computer engineering that doesn't use the word *computer*, and I think computer engineers also design with FPGAs (field-programmable gate arrays) or ASICs (application-specified integrated circuits) that aren't necessarily computers. Despite all this, we aren't going to find a single definition that is correct with other definitions being wrong. The term means different things

to different people and in different contexts.

MADSEN: To me, CE is about design, analysis, and implementation of machines to compute, i.e., computer systems, hence, digital systems as a term for what we do is at the same time way too broad and way too specific.

JOHN M. ACKEN: In a world where the IoT (Internet of Things) includes small measuring devices, I don't think limiting computer engineering to digital values is appropriate. I do think we are talking about digital computers, but computer engineering needs to at latest consider the A/D (analog/digital) concepts for measuring devices.

MANI SRIVASTAVA: Plus one for Robert's simple definition, with the provision that "computer" may be embedded in a system that we don't even think of as a computer system. And most certainly we must not limit to digital or symbolic!

MARILYN WOLF: I will throw in my two cents and thus show my hidden agenda. I believe that computer engineering is not limited to hardware. The old-school definition of CE versus CS is that computer engineering is hardware, computer science is hardware. The typical introductory computer engineering course covers logic design. I still think that the definition I've used for quite some time is useful: *computer engineering deals with time in computing*. Computer science progressed in part by abstracting away time. Digital system design doesn't have the luxury of avoiding time, neither does real-time software.

MADSEN: It is a very important aspect that Marilyn brings up. Having many CS students attending our introductory computer systems course, it is apparent that time is not on their radar, except as time it takes to complete the running of a program. They end

up being better programmers when understanding that computing requires understanding of time and space of bits.

SRIVASTAVA: Back to what is "CE," limiting it to "digital" is shortsighted IMHO (in my honest opinion). Firstly, traditional digital abstractions are being stressed with all sorts of stochasticity and variations at the lower tiers. Secondly, surely things such as analog neural accelerators, spiking circuits, et cetera are surely part of CE. The technologies used in "computers" are evolving and CE evolves with it. Not too long ago, computers interacting tightly with humans and the physical world were outliers, but now they are the norms. So things such CPS (cyber-physical systems), HCI (human-computer interaction), et cetera crept into CE. What I like about Robert's definition is that it naturally adapts to changes in technology and abstraction. The Electrical and Computer Engineering Department at UCLA is relatively recent (2017), and we certainly went with a more modern interpretation of CE and didn't limit it to "digital systems" processing "symbolic information."

ACKEN: I think we need to be careful about emphasizing the distinction of CE. As David mentioned, different places have different definitions, and the definitions change over time. There is a very large overlap of CE with EE (electrical engineering) and CE with CS. As an example, I would expect all three to know about digital logic (analysis and design). However, I would expect all CE and EE students to be familiar with logic circuits. I would expect all three to be familiar with some software programming. However, I would expect all CE and CS students to be familiar with parsing. Some schools link CE and EE, and some link CE and CS. I expect every EE to know Ohm's law well enough to solve complex circuits. I expect every CE to know computer components to

relate instructions to hardware. I expect every CS student to understand algorithm complexity analysis. Of course, any individual may know all three, but as groups, there are some common minimums. A slightly different perspective is to ask the question of what a hiring manager might ask about a degree. In many cases, job postings list all three degrees. Why would a hiring manager pick a candidate with an EE degree over a CE or CS? Why would a hiring manager pick a candidate with a CE degree over an EE or CS?

DICK: The CE grad is guaranteed to know enough about algorithms and implementation substrates to design a complete working system. The EE and CS grads may, but their degree doesn't certify it. They can specialize, e.g., on circuits or theory, and still meet degree requirements. At least that's how it works at UMich (the University of Michigan). The requirements are less flexible than for CS or EE.

SRIVASTAVA: Thinking in terms of layers of abstractions is good, though I wouldn't be so CPU centric! (processing, storage, networking, physical world interaction, human interaction are all part of modern "computer system").

COMPUTER: Robert's definition certainly makes a strong case for CE. Are there limitations to a CE degree compared to the other two? To put it another way, how strong are these components of a computer engineering degree relative to CS or EE:

- › circuits and devices
- › signal processing
- › algorithms
- › software engineering.

BROCKMAN: I like the way that John phrased his expectations for the three degrees, and it's similar to the answer that I give students when they are trying to decide whether to major

in EE, CS, or CE. I also think of CEs as being most expert of the three in register-transfer level design, to take into consideration nonvon Neumann digital processors. I teach the logic design course at Notre Dame with students from all three majors that covers topics from simple switching logic and AND gates up through the design of a simple RISC (reduced-instruction-set computer) processor, and a bit of assembly language and I/O (input-output) interfacing, using Verilog and FPGA boards. I tell students that if you really like this stuff and designing systems that operate on 1s and 0s but aren't as interested in what happens between a one and a zero, you should consider computer engineering. If you are really interested in currents, voltages, and devices, then EE is for you. CS students may have some interest in logic design, but definitely prefer the world of software and algorithms.

With regard to the components that Marilyn listed, in terms of what employers might expect from people with CE, CS, and EE degrees:

- › *Circuits and devices*: Both EE and CE should understand the basics of digital integrated circuits, but truly analog circuits, amplifiers, small signal, et cetera probably not be required for CE.
- › *Signal processing*: This is a tough one. Depending on the focus and level of abstraction, it could be relevant to all three, but the traditional mathematical topics seem to be mostly in the domain of EE. CEs, however, should be prepared to design an accelerator or coprocessor at the RTL (register transfer level) level, even if they don't understand all the theory behind where the coefficients come from.
- › *Algorithms*: Critical for CS, strongly recommended for CE. Typically not part of an EE degree.
- › *Software engineering*: All three majors need to be able to

program in some high-level language. EEs rarely take software courses beyond basic programming. CS and CE definitely need data structures. Both should also have some basic background in OS (operating systems). CE should have some understanding of compiler backend, at least how basic C statements turn into assembly language, so that they can properly "relate instructions to hardware."

WOLF: CAD, of course, is a special case. CAD folks need to know digital logic, circuits, and algorithms.

BYRD: While I think the general distinction between "pure" EE (if there is such a thing) and CE is pretty clear, it gets harder to distinguish with CS. I like to think about CE as "hardware first" and CS as "software first," which sort of reflects the two curricula here at N.C. State (North Carolina State University). Our CE students do a lot of programming and software design, but there are fewer formal courses in software than in CS. It's interesting that Marilyn brought up signal processing. That's considered an EE topic here, and the bulk of our machine learning courses are associated with those faculty. I encourage our CE students to use their electives to learn about signal processing as well as controls to give them more systems-level skills.

WOLF: IoT is an example of computer engineering in the service of signal processing.

BROCKMAN: Even the "pure" EE thing is definitely in flux. There are discussions here about whether all EEs need to have required courses in circuits, devices, or electronics courses if they are headed in the direction of communications/coding theory, control systems, embedded systems, et cetera. Part of this is motivated by what industry is looking for, part of it is motivated by what students are interested in, part

of it is motivated by what faculty want to teach. All very interesting! There is also the distinction between theoretical and experimental work. There are Ph.D. theses in both EE (coding theory, et cetera) and CS (algorithms) where the last sentence is "QED." (I've never been that sure of anything in my life.) I've always thought of CE as pretty much experimental/practice oriented. Labs where you get your hands on real hardware in several of the forms that computers take today seems pretty central to the experience. This takes a serious commitment to resources in terms of space, equipment, and especially experienced lab instructors.

WOLF: Computer engineers, particularly CAD people, may develop algorithms but that is not the end of the story. Experimental validation is required because many of these efforts attempt to find practical solutions to formally intractable problems.

MADSEN: I think that we are missing a very strong aspect of CE, that of systems understanding, that those computer systems which we develop are themselves systems of systems as well as part of other systems. The latter, which do include the human aspects.

WOLF: Computer engineering has evolved from design of computers to application systems built from software and hardware. A smaller fraction of computer engineers now perform VLSI (very large-scale integration) design. A lot of companies design at the board level. That Bell Labs ASIC model has moved onto boards plus FPGAs.

COMPUTER: How well do we train our students for their careers in computer engineering?

SRIVASTAVA: One conclusion I've reached reading the thread is that what we call CE here at UCLA perhaps won't be viewed as CE at the universities represented by others on this conversation: We certainly have the "classical CE" that everyone is talking

about here (very processor and conventional digital system centric) as a pillar, but we also have equally strong pillars on how a system interacts with the physical world and humans, how to design for properties beyond area/timing/power, and an emphasis on application context.

ACKEN: I think Mani has an important point. Specifically, every university is going to have different topics under each heading. So, while we concentrated up to now on what distinguishes the three (CE, EE, and CS), I think Marilyn's question, "How well do we do training our students for their careers in computer engineering?" should also include how well we do in all three fields. For example, there are basic engineering problem skills that apply to all three areas, and they must be included in answering how well we prepare CE students. I just realized in our discussion that for general CE ideas at the BS [bachelor of science] level there is much more overlap with CS and EE than if we are considering a PhD CE.

SRIVASTAVA: I'm not sure how to answer, "How well do we do training our students for their careers in computer engineering?" considering that the job market is skewed so heavily toward software and data science-oriented jobs, and so that is where most students end up irrespective of whether the degree is CE, CS, or even EE. In India, there is a degree called *masters in computer applications*: how to engineer applications. At least they're honest about what the typical students do postdegree! My advice to students tends to be that an MS (master of science) degree is a must for a healthy preparation for a career, unless one is happy being a coder or a Q&A tester, essentially a five-year basic degree.

HARRIS: Regarding training students for their careers, I think most CE/EE/CS graduates are getting good jobs these days and employers are not

complaining loudly that universities are not preparing them adequately. The United States continues to be a leader in the field, so at least a fraction of our collective graduates are extraordinary and many are strong. Clearly the quality of the graduates varies across individuals and schools, but also the preparation required for jobs varies across fields and there is room for different people with different skills. I don't know that the title of the degree is an important filter for

employers. In my roles in industry, my teams have considered applicants based on skills rather than title of degree. I've seen math and physics graduates do very well in jobs that might traditionally be considered computer engineering, and either EE or CS majors could do very well in computer engineering if their interests align. Similarly, software companies hire lots of non-CS majors who can program and tackle hard problems. Many jobs require skills that are too specialized to teach in most undergraduate programs, and many jobs five years from now will require skills that scarcely exist today. We primarily need to produce graduates who have a foundation and mindset to learn new things. On the whole, companies seem to feel engineering graduates are pretty successful at doing this. There aren't a lot of midcareer engineers getting laid off because they are unable to keep up with technology, though this can be a problem at the lower end of skill levels. Harvey Mudd College offers an undergraduate general engineering degree. Grading seniors at our department reception yesterday are going to a wide range of places, including major semiconductor makers, aerospace companies, and software companies. Few of

them have as much computer engineering coursework as a traditional CE major, but they have a lot of experience learning to solve problems in new areas, and good experience working on teams and communicating their work.

WOLF: I agree that an MS is a minimum for a healthy career.

ACKEN: As David said, most CS/EE/CE grads are getting good jobs, but my students have shown a clear distinc-

I encourage our CE students to use their electives to learn about signal processing as well as controls to give them more systems-level skills.

tion in success based upon BS versus MS. All of the MS students in (EE and CE) are getting snapped up fast. Not as fast for the BS. However, it seems the CS students are getting snapped up fast at both the BS and MS levels.

MADSEN: At DTU, the B.Sc. (bachelor of science) (three years) is regarded as step one to get a M.Sc. (bachelor of science) (two years). If you want to stop after three years, you will go for a B.Eng. (bachelor of engineering), which is a more applied engineering degree.

SRIVASTAVA: Fully agree about the focus on long terms; the most important things, I believe, we should seek to impart are

- › how to solve new problems (in computer systems design)
- › how to continually learn (as computing technology and abstractions will change)
- › understand the foundational concepts that are not dependent on short-term technology trends.

DICK: They quickly land positions they are happy with and in interviews

years after graduating are generally happy with their experience. Based on those interviews, I can say that a lot of graduate seldom encounter some of the things I think are fundamental and important in the long-term. Interviews with senior people at companies suggest they are mostly happy with the preparation of our students. However, we end up teaching them enough of the more immediately practical material (that we might not think is really the goal), that the non-research-track students are mostly happy with the experience. A quick summary would be not bad, could be better. Doing a good enough job that we better be careful not to break things that are working when making changes to improve. Research interests naturally pull us toward things that soon have high demand, e.g., we were teaching interested students about low-power WANs (wide area networks) and machine learning before they were hot, although we generally don't require these topics.

BROCKMAN: I'd like to touch on a couple of things that Mani and David brought up: the relationship of computer systems to the broader human-centered application context and engineering education in general. I think we're seeing a similar trend to what David mentioned, that the employers hiring our students across EE/CE/CS aren't as focused on what degree they have but what are their skills and experiences. What seems to be emerging as a differentiator is experiential learning opportunities: students working on complex, real-world projects either as part of coursework, internships, clubs, et cetera. By their nature, these projects are often multidisciplinary. Within the confines of the CSE (computer science engineering) department, the courses I teach have been the traditional CE, processor-centric digital design, VLSI, and computer architecture courses, which is where this thread started. This year, I've worked on a

new interdisciplinary projects course with an engineering designation that counts as a technical elective for all of the departments in the College of Engineering. Some of the projects also had students from other departments on the teams, including finance and architecture. The work that Harvey Mudd has done with their clinic projects has been a great inspiration for this, and we continue to look closely at their model. There are certainly challenges to doing interdisciplinary research in a university, but those hurdles seem to be easier to overcome than getting a large number of undergraduates involved in rigorous, real-world, interdisciplinary projects that count toward their engineering degrees, be they computer engineers or mechanical engineers.

ACKEN: The problem I have with the idea that hiring managers do not focus on the difference between CE/EE/CS is that does not match very many job postings. I have seen very few job postings (once again I am discussing new college graduates, not experienced engineers) that say CE/EE/CS, rather, I see one or two of them.

HARRIS: Even if the requisite listed a specific degree, I've never worked at a place that wouldn't consider a strong resume just because the name of the degree was different. Again, employers I meet and places I've worked want smart people who have a grounding in the fundamentals, learn new things quickly, have a good attitude, work well with others, and communicate well. If it's a senior position, there's also a need for technical depth in a given niche, but even there, narrow experts aren't as valuable as people with flexible problem-solving experience.

MADSEN: When I ask employers what makes the education at DTU unique, they all emphasize the strong and deep mathematical basis they get in the first two semesters, a skillset which enables abstract thinking.

ACKEN: Well, David, does this mean there is no difference or that many employers know they will be hiring multiple people and any one position can be filled by one of the three but the team needs all of them?

HARRIS: I think it means that there are many electrical engineers who are prepared to build software systems and many computer scientists who could do design embedded systems or chips, for example. The variation between individuals is as great as the variation between degrees, so limiting consideration to a single degree excludes well-qualified individuals. EE, CE, CS, math, and physics are all fields that could prepare students to apply principles of science and mathematics to solve problems related to physical and cybersystems. A physicist who builds instrumentation or a computer scientist who fixes old video games for fun might be better equipped to build IoT than a computer engineer who specializes in CAD algorithms.

WOLF: A few weeks ago, I talked with an alum who runs a design house. He thinks that a lot of companies look for students with very specific skills. He prefers to hire productive people, keep them, and train them. Some of the emphasis on specific skills comes from high turnover.

ACKEN: Yes, David, I absolutely agree that the variation among individuals that can do a job far exceeds the individuals that can do the job. The fact is an individual math major might do much better than an electrical engineering major to design a particular circuit. That doesn't change the fact that there is a reason to have different degrees. What preparation would be preferred for a given job for people with approximately equal capability? When we send out our students we want people to see a benefit in them having a particular degree.

BROCKMAN: We hear increasingly from students and recruiters that they

are looking for students with project experience, more so than certain discipline-specific skills. (The exception to this is the huge pressure that our CS and CE students feel to be able to excel in the coding challenges that are part of the interview process for Googles, Amazons, and Facebooks of the world.) They can get this project experience from tech-oriented clubs, but our goal is to help them get that experience as part of their “day job” as students for academic credit, rather than having to find discretionary time to do this as an extracurricular activity on top of their already-overloaded schedules.

WOLF: This could lead into an even broader discussion about degrees. In particular, have the undergraduate engineering degree distinctions outlived their usefulness? A lot of modern systems combine mechanical, electrical, and software components. An argument could be made that they shouldn't specialize until graduate studies.

HARRIS: John, coming from a general engineering program, I suppose I'm less attached to the importance of particular degrees. I agree with you that a major does ensure students have been exposed to a minimal set of topics; EEs should have seen circuits and differential equations, and CS majors should have seen object-oriented programming and discrete math. I find the number of students at Mudd who pick a major to optimize preparation for a given job title is fairly small. A lot of students pick a major because of an inspirational teacher in a first course, or because of advice from family or friends, or because of perceptions of how hard the degree is or how much money they'll make when graduating. In my experience, most students get a first job in specialty that they didn't know much about when they were entering college, and most engineers five-years out are doing something they wouldn't have expected at the time they graduated. Like Marilyn and

Jay said, people who are generally productive and can tackle new and multidisciplinary problems and work with teams will do well. Many students get to take an upper-division elective, capstone project, research experience, or internship that they fall in love with, and then they seek a first job in that field. Often these experiences are accessible to students from more than one major. I think an important part of education is to give students an opportunity to sample experiential learning in a variety of areas to find at least one where they want to pursue a first job. Overall, I think a great program should give students a broad grounding in the fundamentals of math and science; introduce them to the lasting principles of a discipline; give them experience solving problems in the lab and in the field; expose them to depth in a few areas of their preference; give them practice with teamwork, leadership, and communication; encourage and facilitate them to own hard problems through capstones, internships, research, or project courses; and engage them through the humanities to be well-rounded citizens and critical thinkers. A major gives a coherent theme to the breadth and depth and a community of learners, which are both important and motivational, but shouldn't pigeonhole what the student can later do. This is all pretty straightforward for students who are well prepared, highly motivated, and have the luxury of focusing on their studies. There's another set of students who may have personal and family commitments that limit how much time they can devote to education, or who have weak preparation in math or critical thinking that makes it more difficult to learn the engineering practices. It's an interesting question of how to shape educational programs with the objective of preparing these students in a way that maximizes graduates' opportunities/unit of effort invested by students. It's also an important question of how to achieve the greatest good for the greatest number on a tight budget. When is broad preparation best and under what

circumstances is it better to teach students a specialized set of marketable skills with a smaller investment of student and faculty time?

COMPUTER: We have spent some time discussing the success of our curricula from an economic/career perspective. From a purely intellectual point of view, are we teaching students what they need to know?

BYRD: I'm happy with what we teach our students, but there's always room for improvement. We are trying to figure out how to expose all CE and EE students to machine learning in a meaningful way, both to demystify it and to give an appreciation of where it can be used as a tool. Also, there's often feedback from employers to provide a stronger systems perspective; as has been mentioned, they seem to appreciate breadth in topics, especially with an understanding of tradeoffs and interdependencies. We've considered a unifying platform/framework/project that can be used for a given cohort of students as a multidisciplinary way to make connections, but I worry about buy-in from faculty and the need to keep refreshing the project to avoid piggybacking from one cohort to the next. The VIP (vertically integrated project) approach used at a number of campuses can get at this for limited group of students.

BROCKMAN: We met with the VIP team at Georgia Tech (the Georgia Institute of Technology) and borrowed some ideas from them for our college-wide EG (engineering) course at ND (Notre Dame), which is called *Industry and Community-Based Innovation Projects*. Like the Harvey Mudd clinic program, our projects involve working with external partners, which can be industry, the local public works department, national labs, et cetera. Just today, I got a list of project ideas from the City of South Bend Department of Sustainability related to their strategic plan for reducing emissions that could involve students from every

engineering major and other majors as well. Each of these project ideas will last multiple years, and the trick is defining pieces of the project that can be completed during the semester course or full-time summer internships and moving cohorts of students through these without losing momentum. We are working with campus organizations that are chartered to do this kind of thing. One organization is the Center for Civic Innovation (CCI), of which I am the director, that works on long-term projects in mostly the public sphere. Another is Industry Labs, which was specifically created to help transition the region from a legacy

HARRIS: According to a 2013 study, U.S. engineering graduate rates have hovered around 50% for the past 60 years.¹ According to a 2017 ASEE (American Society for Engineering Education) study, the four-year graduation rates are about 30% and the six-year rates are 55–60%. Engineering is not a four-year degree at most schools. The rates are 10–20% lower for black and Hispanic students, which reinforces societal problems.² But according to [univstats.com](https://www.univstats.com), the average graduation rate of the best engineering schools is 89.17%, and MIT (the Massachusetts of Technology) achieves 95.58% (<https://www.univstats.com/comparison/engineering/graduation-rate/>). I think

ACKEN: As we think about CE education preparing the students, I was distracted by some of the other things we do for students. Here is an interesting article from *The Chronicle of Higher Education* about recommendation letters:³ My last idea is what employers expect from EE/CE/CS degrees. What do we expect? Suppose you are advising a student who asks you whether they should go into CS, EE, or CE. I have had many students ask this. I usually tell them to follow their interests because if they are interested in a subject they are more likely to succeed working on that subject. Some students ask which degree is the most likely to get them a job, and I respond there are good jobs in all three areas. What do we expect from our senior and M.S. students to have as the basics for a CE student? How do we prepare them to meet those expectations?

It's hard to interface embedded systems to the real world without some understanding of analog phenomenon in time and frequency.

“Manufacturing 1.0” economy to “Manufacturing 4.0.” Our team at CCI helps provide the logistical support for the EG course and also runs a summer internship program that employs 50–60 college and high school students each year, so we are able to maintain some continuity on projects year round. Regarding college/department/faculty buy-in, the key thing for us was getting the engineering course approved as a tech elective, which is a lower threshold than a departmental, major-specific elective.

MADSEN: We are expanding our curricula across all technical disciplines, with mandatory courses on innovation and on sustainability in the M.Sc. programs, and have expanded our polytechnical foundation in the B.Sc. to include bioengineering, programming, and statistics [including some ML (machine learning)], besides the classical, math, physics, and chemistry.

COMPUTER: Let's consider some ambitious but achievable change. What would be first on that list for computer engineering curricula?

the most important thing we could do for society is to raise the graduation rates while also raising the quality of education. Accepting students to college but then not graduating them is a tremendous drain on resources. It creates a pool of young people who have invested time and borrowed money but have nothing to show for it. To get here, I think we need to consider metrics about graduation rates, career outcomes, student debt, student satisfaction, and individual faculty teaching quality as part of funding formulas. At programs with low graduation rates, we need to shift emphasis from research productivity to serving students better.

WOLF: While quite a few degree-agnostic techniques have been developed to improve retention, I think that some CE-specific methods might also be attractive.

COMPUTER: As one example of shifts in CE curricula, VLSI played a central role for a long time but seems to have become much less important over the past few years. Thoughts?

WOLF: On the upper-division side, I think that more education in software system design would be useful. Many embedded and cyber-physical systems have tens or hundreds of millions of lines of code.

ACKEN: I was thinking about what is special about CE, with the thought of what every one of these majors should know (what is common) and what one might not be expected to know. Do we agree that all three should have the basics of math and physics? (calculus, linear algebra, classical physics). I think all three should have some knowledge of embedded systems (even if from different perspectives). On the other hand, many EE majors and many CS majors would not have VLSI design, but I would expect CEs to have some VLSI design. Thoughts on whether it is a benefit to identify what topics a CE would be expected to know versus the others?

HARRIS: We've wrestled with these questions after a recent college core change. Electricity and magnetism is no longer required of all students, and engineering would have to give up a

major class to add it back into the major. We are experimenting with not requiring it. Electricity and magnetism was so focused on fields that we had to teach circuits from ground zero anyway. The core also lost some math. We are bumping the engineering requirement up to four semesters: calculus, linear algebra, probability and statistics and differential equations. There's widespread faculty agreement that math is essential. As a VLSI textbook author, I think VLSI is now elective material for computer engineers. The set of jobs that need that knowledge have become a niche, even if it is a niche close to my heart. Embedded systems and FPGAs are much larger job markets.

COMPUTER: Math and physics are important. Exactly how much is an interesting question. How much abstract mathematics do our students need to know to become effective professionals?

HARRIS: I think a large majority of effective professionals know very little abstract mathematics, but the best ones can still use applied mathematics. I think one of the ways a computer engineer may be distinguished from a computer scientist is by having at least a first course in passive circuits and a first course in time and frequency domain. It's hard to interface embedded systems to the real world without some understanding of analog phenomenon in time and frequency. These courses require some knowledge of first- and second-order differential equations as well as some linear algebra. In contrast, the computer scientist needs linear algebra but not differential equations. Discrete math and probability and statistics are relevant to both fields, but mathematical preparation hasn't always kept up with the skills needed for machine learning, and programs are under strain about how much math can be required. Realistically, many of our students dump most of the math from their brains pretty quickly, although they can relearn it in context

faster the second time if they need to apply it for a course.

WOLF: Two reasons are given to teach math to engineering students: specific mathematical knowledge and mathematical maturity. What types of math provide the best foundation for the mathematics used in computer engineering? And perhaps we should enumerate the types of math important to the field: differential equations, linear algebra, mathematical logic, discrete math.

MADSEN: I already mentioned how we are dealing with these foundational topics. We are currently linking the math and programming courses by introducing discrete math and basic programming into the math course. We are also introducing computational thinking across math and programming.

BYRD: I like David's summary of the math. I think that statistics is an area that could use more attention; professionals need to do performance analysis, usually with experimental data, and they need to be able to know whether they are drawing reasonable conclusions from the data.

DICK: Computer engineering is special in its emphasis on discrete math, logic, and, to a lesser degree, combinatorics with the thought of what every one of these majors should know (what is common) and what one might not be expected to know. Do we agree that all three should have the basics of math and physics? (calculus, linear algebra, classical physics). There are some arguably successful programs that do not require linear algebra, instead opting for vector calculus. However, this is most likely for consistency among engineering math requirements. The trend is toward requiring linear algebra. I think all three should have some knowledge of embedded systems (even if from different perspectives) On the other hand, Many EE majors and many CS majors would not have

VLSI design, but I would expect CEs to have some VLSI design. Thoughts on whether it is a benefit to identify what topics a CE would be expected to know versus the others? Here at Michigan, they are required to have knowledge of computer system implementation, including hardware. However, they are not required to take VLSI design, architecture, and embedded systems. They get basic exposure to all three in other courses, but it is fairly common for students to graduate without any dedicated VLSI course. We have noticed student demand for embedded systems and robotics growing faster than for general-purpose microarchitecture and VLSI design. Permitting specialization here might partially be due to each of the three courses (VLSI design, architecture, and embedded systems) taking 35–40 h/week on average. Requiring all three would make the major inaccessible to many competent students.

ACKEN: I agree that most professionals don't use their basic math skills very often. However, I do believe the skills are the necessary basics for the studies the students will use. This, I think, is important for an engineer to be able to know the boundaries of application for a particular solution. The practicing engineer doesn't need to be able to rederive the underlying equations the simulator uses, but they do need to understand the limitations of applying the simulator models. Therefore, while the student learns engineering specifics in upper-division classes, they use the basic or applied math to really learn those concepts rather than just a fuzzy set of rules or a long list of equations.

BROCKMAN: There's a general consensus that math and science are core components of a college education, not only for engineers but for all majors. This is only partly because students might use these tools on the job but more so because of the habits of mind that they promote and how they help us

see the world around us. A key question is what math and what science should students—regardless of major—be studying today? More than 300 years later, our college math/science core is still heavily based on the work of Galileo, Newton, Leibniz, Huygens, Boyle, Hooke, et al. This is the foundation of classical mechanics, which in turn gave rise to the machines of the industrial revolution. Some understanding of the math and science that brought us the first Industrial Revolution is essential to understanding the world as it is today. But arguably, no machine is having a great impact on our lives today than the computer, and it is time for higher education to catch up to that. It's time to start thinking about dialing back on how much calculus and continuous math we require and replacing it with discrete math, not only for the education of the professionals who will be developing new computing technologies but also for the general public who will live with its consequences. History will show that Claude Shannon was as important as Isaac Newton.

COMPUTER: Are there any curricular topics on the chopping block? Any topics we should get rid of, both to avoid clutter and to make room for new topics?

ACKEN: Some topics that CE would benefit but we could let them choose among a set of prereqs (prerequisites). All should have basic circuits and digital logic, but some special choices: VLSI circuits, analog circuits, board-level design, I/O interfacing. All should have a basic computer architecture class but some choices: advanced computer architecture, cache analysis and design, special adders and multipliers, and parallel processing (vector processors and GPUs).

HARRIS: I agree that computer engineers could have a single course that covers practical interfacing circuits in the time and frequency domains, such as how to hook up switches, LEDs (light-emitting diodes), motors, and various resistive and capacitive

sensors, along with how to build an op-amp antialiasing filter to go before the ADC (analog-to-digital converter) on the microcontroller. We do some of this in our sophomore lab for all engineering majors of all types and are debating whether to go further in this direction. I'd strongly argue against the idea that engineers don't need to be able to analyze basic first- and second-order circuits. I've worked with engineers who only know how to use the circuit simulator; they have no idea of what parameter to tweak to make their circuit better or why, when to stop tweaking, and what the ultimate limits of performance of their topology would be. More importantly, simulations are usually wrong the first time they are run, and without a good way to predict the answer, inexperienced engineers believe the pretty picture from the simulator and report false conclusions. I proposed a single (semester) course for computer engineers. It would be a full year if there were one semester on circuits and another on signals and systems.

BROCKMAN: One of the main things that I'll want to comment on is that computer engineering programs seem to exist on a continuum between EE and CS, and what we require of students can vary, especially depending on whether CE is in the same department as EE or CS. There is a lot of material to consider along this spectrum and we can't possibly fit everything from electrons and holes to algorithms into nominally four years. ND is a particularly interesting case: the Department of Computer Science and Engineering was spun off from the Department of Electrical Engineering in 1991 when the university decided to create a computer science program within the College of Engineering. (I was one of the first new hires as an assistant professor in 1992.) One of the distinguishing features of our program is that we have generally kept a common course sequence through sophomore year for CS and CE. When the CSE department

was first created, there was a fairly large number of EE majors, and in the early years of CSE, the number of CS, CE, and EE majors was comparable. Today, CS is around 70% of the three majors, EE 20%, and CE 10%. The logic design course that I teach is required of all three majors, and as the distribution of students has shifted, the course has evolved a great deal. Originally, the course was much more EE-ish, today it is much more CS-ish.

COMPUTER: Perhaps we can identify the centroid of computer engineering versus computer science versus software engineering as a way to tie together the ideas we have discussed here. The Venn diagram of these fields overlap significantly. We can also look at these skill sets as statistical distributions.

ALL: Where is the centroid?


- ▶ CS centers on data structures. Core courses: algorithms, object oriented-ish programming.
- ▶ EE is centered on devices; systems science. Core courses: currents and voltages, transforms.
- ▶ CE focuses on the physical costs of computing. Core courses: logic design, architecture.

ACKEN: VLSI is less at the center than it used to be, although the number of design starts is going up, although majority of students are system designers. CE is often the least flexible major due to the number of topics. Computer engineering must be distinct from EE, CS. Design computer systems, design systems using computers. Many programs have few or no courses that are unique to CE. Alternative approach is a build-your-own degree. General engineering degree is yet another model.

HARRIS: The processor is centroid of computer engineering. The centroid has overlaps with other disciplines. Somebody needs to lead the design of digital hardware. Somebody needs to design data centers. Somebody needs to

design embedded software. The world needs these people. Any given society may be a producer or a consumer of these fields. CEs should be able to write software; design a pipelined processor design an embedded system. Icing on the cake is systems engineering.

BROCKMAN: Given a problem: CS centroid is software, EE centroid is hardware, CE centroid is tradeoff. These fields share many common tools, a given person may gravitate toward one set of tools over another based on the field in which they were trained.

COMPUTER: Thank you, everyone, for this great discussion! 

REFERENCES

1. B. N. Geisinger and D. Raj Raman, "Why they leave: Understanding

- student attrition from engineering majors," *Int. J. Eng. Educ.*, vol. 29, no. 4, pp. 914–925, Mar. 2013. [Online]. Available: <https://www.rise.hs.iastate.edu/projects/CBiRC/IJEE-WhyTheyLeave.pdf>
2. "Engineering by the numbers: ASEE retention and time-to-graduation benchmarks for Undergraduate Engineering Schools, Departments and Programs," American Society for Engineering Education, Washington, DC, USA, 2017. [Online]. Available: <https://ira.asee.org/wp-content/uploads/2017/07/2017-Engineering-by-the-Numbers-3.pdf>
3. B. Schreier, "No More Letters of Recommendation!: This hyperstylized, dishonest genre is useless for everyone," *The Chronicle of Higher Education*, Washington, DC, USA, May 2022. [Online]. Available:

https://www.chronicle.com/article/no-more-letters-of-recommendation?utm_source=Iterable&utm_medium=email&utm_campaign=campaign_4237995_nl_Academe-Today_date_20220510&cid=at&source=&sourceid=&cid2=gen_login_refresh

MARILYN WOLF is the Elmer E. Koch Professor of Engineering and Director of the School of Computing at the University of Nebraska—Lincoln, Lincoln, NE 68588 USA. Contact her at mwolf@unl.edu.



IEEE TRANSACTIONS ON BIG DATA

► **SUBSCRIBE AND SUBMIT**

For more information on paper submission, featured articles, calls for papers, and subscription links visit: www.computer.org/tbd

TBD is financially cosponsored by IEEE Computer Society, IEEE Communications Society, IEEE Computational Intelligence Society, IEEE Sensors Council, IEEE Consumer Electronics Society, IEEE Signal Processing Society, IEEE Systems, Man & Cybernetics Society, IEEE Systems Council, and IEEE Vehicular Technology Society

TBD is technically cosponsored by IEEE Control Systems Society, IEEE Photonics Society, IEEE Engineering in Medicine & Biology Society, IEEE Power & Energy Society, and IEEE Biometrics Council

