Algorithms, data structures, OSs, database design, compiler design, and programming languages were once the core ingredients of computer science (CS) education—until universities ignited the computer technology revolution by producing the inventors of Yahoo, Google, Facebook, and others. So, is commercial technology forcing CS curriculums to adapt, or are curriculums so rigid that they ignore various trends? CS graduates were once prized for their ability to generate accurate, actionable information. In a time where misinformation and disinformation run rampant, where do today’s core CS educational ingredients fit in?

Are we producing CS graduates who understand the core principles that fuel technologies such as IoT, cloud, blockchain, mobile apps, “fake news,” data analytics, and big data? And do CS faculty understand the complexities of the security and privacy challenges associated with these trends? Or are our educators teaching the same CS 101 classes from 40 years ago with little refresh?

We surveyed six of the best senior CS educators: Phillip A. Laplante, Michael Lewis, Keith Miller, Jeff Offutt, Jon George Rokne, and Shiuhpyng Shieh. The panelists’ individual insights are presented below. (See the sidebar for more information about the panel.)

COMPUTER: What role can CS education play regarding the daily misinformation and disinformation provided to the public? Are CS curriculums adapting at the speed of change experienced by the technology and media sectors?

SHIUHPYNG SHIEH: Misinformation can be distributed in any form of media; the Internet is the cheapest and fastest way. The misleading information can be controlled by in-depth analysis using natural language processing or rule-based filtering. With these techniques, inappropriate content for children has been blocked or removed by Facebook and Google to provide a healthy environment. However, CS has been expanding too quickly for the curriculums.

JON GEORGE ROKNE: The daily occurrence of misinformation and disinformation experienced today is, to a large extent, a social problem made possible by the introduction of electronic communications over the Internet, a CS artifact. Because the proliferation of questionable information is a social phenomenon, it is not a problem that can be solved by CS alone. Nonetheless, CS as a field might, through the use of AI techniques, provide some relief from...
the proliferation of false information by flagging incorrect information. Robustness was the main consideration when the electronic communications systems that evolved into the Internet were designed. Hence, we now have a communications network with only limited control over the originator of a statement (no matter how egregious) to be accountable for its veracity.

**MICHAEL LEWIS:** CS education does not have a specific role in dealing with daily occurrences of misinformation beyond being part of a college curriculum that should instill its students with a healthy skepticism and the understanding that what they are hearing at any particular moment is likely not the whole truth. Proposals for a role for CS in dealing with misinformation and disinformation are alarming. Attempts to algorithmically police the Internet and purge it of objectionable content would likely become an enforcement of the biases and tastes of those doing the policing. CS curricula are not changing as fast as the technology and media sectors. Structurally it would be difficult for them to do so.

**KEITH MILLER:** I think CS education can play a role in sensitizing students to their professional responsibilities with respect to information, misinformation, and disinformation. I also think CS education should play such a role. However, I am not particularly confident that CS education is doing a great job of this at the moment. But when we examine the curricula, we do see some progress in including more professional ethics and social issues.

**PHILLIP LAPLANTE:** I think CS education can play a role in making the public aware of privacy and security vulnerabilities and in diffusing misinformation, and I think, for the most part, they are doing that. I think the greatest area of disinformation is in AI, where we often hear in the mainstream press about self-aware computers and robots rebelling against their masters and taking over the world. I have heard even knowledgeable CS graduates spread this fear. The removal of certain topics from the core curriculum might have created a class of CS graduates who are excellent tool users but don’t fully understand what is happening under the covers. This lack of understanding, in turn, leads to a fundamental ignorance of certain realities—what AI really is, security awareness, and so on—leading to the propagation of disinformation.

**JEFF OFFUTT:** Bear in mind that people have been creating disinformation for thousands of years. American-style racism is a great example. It convinced poor white people that black people wanted to escape slavery to take their jobs. That misinformation campaign took half a century to disseminate, but networks and social media allow incorrect information to propagate to more people faster. So, CS has to accept some responsibility for enabling the massive spread of incorrect information. But it’s not a software problem, and I doubt if it has a software solution. Maybe the solution is to teach critical thinking and analytical skills. How do you and I know which information is valid and which is false? How are we, as CS educators, adapting? Not so well. CS education is very conservative; currently, our biggest problem is keeping up with the sort of suspicion sometimes aimed, say, at the chemical industry or big agriculture. Nor are the tech sectors’ products viewed with suspicion—consider how people react to smart watches as opposed to pesticides. This is a bit surprising given the numerous massive privacy and security breaches that have been in the news. What magnitude of disaster would it take to turn people against algorithms and computers?

**MILLER:** My general sense is that most people think of us as geeks, a cross between technicians and high priests of technology. However, events such as the Volkswagen diesel cheat (R. Oldenkamp, R. van Zelm, and M.A.J. Huijbregts, “Valuing the Human Health Damage Caused by the Fraud of Volkswagen,” *Environmental Pollution*, vol. 212, 2016, pp. 121–127) might make
the public more suspicious of the people behind the machines.

**OFFUTT:** I loved the book 1984 when I read it in the mid-1970s, but it terrified me. Go ask 100 young, educated, Chinese people how well the “great firewall” works and whether they use a VPN. I guarantee that every single one will say, “What’s a VPN?” because they know if they admit to using it, then they will have an unpleasant conversation with the local security forces.

**SHIEH:** People do not trust beyond their understanding. System security cannot rely on opaque design.

But they know what VPNs are, they use VPNs, and they view the great Chinese firewall as a porous fishing net. The dystopian society described in 1984 is not possible because the tools the politicians use to keep us uninformed are the same tools we can use to learn more than our parents ever imagined.

**LAPLANTE:** I think when discussions about security, privacy, and so on are not grounded in mathematical theory, then they are not very meaningful. But when discussions contain the requisite mathematical theory, then this might create mistrust in those who do not understand the theory or who dismiss it out of hand, especially if the theory refutes their position. But theory can be used both to inform and as a weapon. By that I mean that one can sometimes massage assumptions to make the theory bear out a certain result; then you unduly claim victory because you have theory on your side.

**ROKNE:** Security, privacy, secrecy, and surveillance are overlapping topics. These terms are also not precisely defined. In CS and mathematics contexts, they tend to be discussed as separate topics. Wikipedia (en.wikipedia.org/wiki/Security) offers the following:

> Security is the degree of resistance to, or protection from, harm. It applies to any vulnerable and/or valuable asset, such as a person, dwelling, community, item, nation, or organization.

In this definition, resistance and protection from harm can often be accomplished using electronic protection devices and communications (connections to security companies). However, communications technologies have enabled widespread installation of surveillance equipment connected to data collection. For privacy and security, the use of encryption and obfuscation for computer communications has become a necessity. Communications that cannot be trusted create a society where everything is suspect. Claims of transparency are met with the same suspicion. In the past, publication in an official medium such as a newspaper was generally held to be correct. Publication of false information had negative consequences for the originator and the publisher. These consequences generally do not exist for online media. Sadly, the generalized mistrust of electronic media carries over to traditional respected media—gone are the days of “If you see it in The Sun, it’s so.”

**COMPUTER:** Sophisticated algorithms can generate fake paper submissions that get accepted to conferences. At the same time, news organizations have begun using AI systems to generate articles on routine topics such as sports scores. Together, these developments would suggest that more advanced algorithms can also generate “believable” false information. Is this premise true? And if so, do you know if it is in use?

**MILLER:** Clearly, many supposedly advanced algorithms are already generating false information, some of it notoriously obnoxious. For more on this, see Aylin Caliskan and her colleagues’ work “Semantics Derived Automatically from Language Corpora Contain Human-like Biases” (Science, vol. 356, no. 6334, 2017, pp. 183–186.) Also, a 2016 article by Emiliano Ferrara and his colleagues described several bad-acting social bots (“The Rise of Social Bots,” Comm. ACM, vol. 59, no. 7, 2016, pp. 96–104.)

**SHIEH:** Although current AI techniques can generate much believable information, the fake information is still detectable by experts or people with high awareness. However, with the rapid evolution of AI, machines are likely to have ability to deceive humanity in the future.

**LAPLANTE:** The fake papers that have notoriously been accepted to conferences were clearly nonsense upon even a cursory review. These papers were accepted because no review was done by the conference organizers. In a few years, we will likely reach a point where a sophisticated program could generate a research paper that could pass a cursory review. But, I think we are a long way off whereby a research paper could be generated that could only be proven to be fake by a careful, expert review.

**ROKNE:** The refereeing process is inexact and dependent on subjective
evaluations of papers. Excellent papers and results are sometimes rejected, and substandard work is sometimes accepted. If a paper generated by a sophisticated computer program is refereed in a substandard refereeing process, it might well be accepted by some publication.

OFFUTT: I believe that if the reviewers, conference program chairs, and journal editors apply due diligence and critical thinking, they should be able to identify fake papers. However, there is a movement toward anonymizing the authors of paper submissions. Although this has some benefits, it could make identifying fake papers more difficult. I'm personally more worried about plagiarism, guest authors, ghost authors, and made-up data. I think this is far more likely. We have tools that detect plagiarism by comparing submitted papers with thousands of published papers. If an algorithm can generate a fake paper, couldn't another algorithm differentiate human-written papers from computer-written papers?

LEWIS: Humans can generate believable false information, so it’s not clear what is to be gained by automation, unless one wants to rewrite history on a grand scale. The more pernicious influence of algorithms lies in web search engines. The web has made us lazier, and we’re inclined to accept as authoritative the first two or three hits returned in a web search. The other big impact of computers on believable false information is that they can disseminate it to self-selected audiences who are already inclined to believe it.

COMPUTER: Have third-party components affected how software architecture and design are taught? Since third-party components are black boxes, there are usually assumptions that do not hold true. Are composability and interoperability taught as core principles in CS education? Or are they considered to be topics belonging more to systems engineering?

ROKNE: Third-party components and programs are used in most CS courses, at minimum to compute examples. One rough test for third-party software is to compute a few examples for which solutions are known. Failure to compute reasonable solutions to these examples would indicate that the software was not robust. Composability and interoperability are generally not taught as separate subjects. When taught separately, they are usually part of a systems engineering curriculum.

SHIEH: For cost consideration, developers usually do not examine the third-party software components. OpenSSL, a commonly used library for securing communication, was found vulnerable. The Heartbleed bug in OpenSSL affected thousands of products. As we can see, the cost of security problems can be unaffordable. The composability and interoperability issues have been covered more or less in CS courses such as computer security and software engineering.

LAPLANTE: I don’t see anything specific in CS curricula about third-party (black box) components. But when the core discrete math topics are taught, these can be used to teach the theory of interoperability, composability, testability, and so on. The availability of industrial-strength open-source software changed the way I taught almost 20 years ago. For example, in testing courses, you can test real software from open-source repositories. Open-source software should be used for “dissection” to study software architectural and design patterns, and for studyingilities such as usability, maintainability, and understandability.

LEWIS: Composability and interoperability are at the very least implicit in how we teach students to design and program. Whether they are enunciated explicitly is another matter—I suspect many CS faculty lack formal training in software engineering and system architecting.

MILLER: Many good programs teach their students to be careful about their assumptions when using software as a black box. And furthermore, good programs encourage students to document their own assumptions and limitations when writing software used by others. The availability of software packages (for example, for graphics and for webpage formatting) has changed how software development is practiced and taught. We might not emphasize interoperability as much as we should, but we do cover it.

COMPUTER: The demand for employee candidates with hands-on experience is growing. Some more senior CS faculty have had little hands-on experience for decades. Has the role of graduate students as instructors impacted CS education in any way?

LAPLANTE: The best football coaches often haven’t played at a high level (college or professional) in many years. But that doesn’t matter because there is so much to be learned from the knowledge of the coach. I haven’t written industrial-strength software in more than 25 years. But I can still teach my students from my past experience, from what I have learned since, and in relating the research literature to the real world. Still, there is tremendous value in fresh, real-world experience, and every time I can, I seek to get that through my consulting and sabbaticals.

MILLER: It is important that many senior CS faculty haven’t programmed anything for decades. It isn’t necessarily crippling when teaching some of the curriculum, but it does erode students’ confidence if a professor clearly doesn’t know what he or she is talking about. Having graduate students as instructors might help that situation, but the graduate student might not have other intellectual tools important for teaching. The same is true for adjuncts who have recent experience: they might have insights that can be
particularly useful for students, but they also might need some coaching to be effective teachers.

**SHIEH:** It is important to balance the roles of theory and practice. Many software and network security problems arise due to inappropriate implementation rather than theoretical issues. Teaching assistants usually have more hands-on experience about system cracking, bug patching, and vulnerability discovery. These ethical hackers bring vivid materials into classes. In this case, graduate students as the instructors or teaching assistants will play a critical role in instructing software development and leading the lab work.

**LEWIS:** I tend to think that graduate students as instructors are not that different from faculty. Departmental practice and the structure of courses tend to dictate what goes on in the classroom, particularly in the lower-level classes that graduate students tend to teach.

**ROKNE:** The role of graduate students as instructors does not impact CS education to a large extent, assuming the students are capable instructors. Graduate students work with their professors, and their conversations form ongoing tutorials. Teaching is the practice of learning, and when graduate students instruct, they carry forward the wisdom of their supervisors. There are several well-known factors inhibiting so-called real-world experiences for CS students. For example, the rate of change coupled with the rapidity of redundancy and the fact that the change is driven by ever-changing needs. To respond to this requirement, many universities have put programs in place that provide internship experiences for students. It is difficult to provide hand-on experience within academia because CS faculties might not have much experience of the world outside academia. Supervising internship students mitigates this problem to some extent.

**OFFUTT:** It is unfortunately true that many CS faculty have little or no experience or understanding for how software development actually works. This is reflected in the lack of teamwork in courses, overly heavy emphasis on theory over practice, and a one-and-done emphasis on grading instead of a learn-and-revise emphasis on learning. Most graduate teaching assistants are PhD students who primarily focus on their research. Teaching is viewed as unfortunate grunt work necessary to pay the bills. I don’t see many graduate teaching assistants encouraging more hands-on experience for their students. Our undergraduate concentration in software engineering at George Mason requires students to have an internship at a software company. In addition, we strongly encourage our students to get involved with open-source projects to get demonstrable hands-on experience. We also are increasingly using undergraduate teaching assistants to apply peer instruction to improve student learning. Among other benefits, our undergraduate teaching assistants often encourage their younger colleagues to get hands-on experience. Thus, I find that our undergraduate peer instructors encourage more hands-on experience than faculty or graduate students.

**COMPUTER:** In past years, issues such as memory allocation and using as few bytes as possible were important parts of CS education. Then, “twiddling bits” was a necessity. But today, with access to clouds and seemingly infinite memory and storage, bytes might be rarely mentioned. Do you think today’s CS students understand such foundations given that today time and space appear infinite? Or, does the sudden proliferation of tiny Internet of Things (IoT) devices suggest a new focus on managing scarce computing resources? What computing limitations do students believe in today, if any?

**LEWIS:** In my experience, today’s CS students have a much more difficult time understanding the need for resource management than in the past. This is odd, too, since they have vastly greater personal experience with software and systems behaving badly. Perhaps, in time, experience with programming the IoT will change their perspective. The power of modern processors and the availability of large amounts of memory and storage are definitely the main reasons students are not mindful of resource management. The use of kinder, gentler languages such as Python in our introductory programming courses also abstracts away concerns about bits and bytes. We’ve noticed in recent years that computer organization is the core course most difficult for students to “get.” Cellphones are also changing students’ abstraction of computers. My colleagues and I have noticed that a surprising number of beginning students do not have a firm understanding of files and how storage is organized, and how data is written to disk. We attribute this to the fact that their most frequently used computing device, a cellphone, hides file structure and app data from the user.

**LAPLANTE:** My experience is that most CS graduates are not exposed to these kinds of problems. But, yes with the increase of tiny and low-power devices in small embedded environments and in the IoT, the problems associated with the very little memory and simple processors are relevant again. In some cases, researchers and practitioners seem to be rediscovering old solutions. For example, much of the work on “low-power software engineering” is simply a return to the compiler optimizations we worked on 30 years ago. And “new” solutions in OSs that can operate on a small footprint look like the real-time kernels I built to run in a 64K memory space for the Space Shuttle 30 years ago. So, many of the lessons we learned back then are becoming relevant again.

**SHIEH:** Nowadays, the significant amount of computer resources lower
the barrier of programming, and consequently, memory and storage are no longer the main concerns for many program developers. However, computing tweak techniques are still very important in some specific, new applications, for example, IoT, big data, cloud computing, and GPU-acceleration. The IoT evolves from embedded systems and wireless sensor networks where resource management remains a critical issue and will not fade away. The modern computing architecture is a layered structure in which some layers manage hardware resources; some implement functions; some focus on user experience. Programming in each layer needs to deal with different limitations.

ROKNE: I do not believe that students are so naive as to believe that resources are available without limit. Students understand that real-life problems still require consideration of speed and memory usage because real problems often deal with large volumes of data that require significant computing power. They are fully cognizant of limitations that will occur in their professional capacities when required to solve real-world problems. The sudden proliferation of IoT devices creates a number of problems, not the least of which are security and privacy. In the general case where IoT devices do not have any degree of local intelligence, there will be a significant new demand on computing resources, and the managing of these will definitely require new and innovative solutions.

MILLER: I find it hard to generalize. The many students interested in programming in the small (for example, using Raspberry Pis or smartphones) are often acutely aware of size limitations. But most undergraduates are far less aware of such concerns, especially when professors tend to give the students relatively small projects. So it depends on what systems and projects the student has worked on by the time they get to me. If you pushed me, I'd say that students today are less likely to think about memory than students were in the 1980s, but students today learn quickly when they bump into limits.

OFFUTT: I’m not sure if bit twiddling is very relevant today. Frankly, I have some issues with modern undergraduate CS education. Why do we educate thousands of students in CS when most become software engineers? My daughter took two years of physics and math to become a civil engineer, and the rest of her courses were engineering. Don’t get me wrong—bit twiddling is probably important knowledge for computer scientists. But the world needs orders of magnitude more software engineers, database engineers, network engineers, security engineers, and robotics engineers than it needs computer scientists. Why do so many students drop out of CS? I believe one reason is that they really want to learn how to engineer mobile apps, not twiddle bits; they want to learn how to organize data with XML, not automata theory; they want to solve problems, not design elegant algorithms. I believe we are on the cusp of a revolution in how we educate students. Some of my colleagues fear that CS will lose, but math didn’t lose when it spawned the natural sciences, and physics didn’t lose when it spawned engineering. Let mathematicians be mathematicians, let scientists be scientists, and let software engineers be software engineers.

SHIEH: Apparently, students are not well prepared for this. This is mainly due to the fact that high assurance stands for high cost. Most commercial products do not need to meet the requirement of high assurance; time-to-market instead often has the priority. Poor software quality is a direct result of expense consideration. However, poor-quality software draws attackers’ attention. The cost for fixing vulnerabilities is often higher than expected. Complete and comprehensive testing is necessary for developing high-assurance software and systems.

MILLER: I don’t think that most undergraduate students are well prepared for working on high-assurance systems. There is not, in my opinion, enough emphasis on testing or quality assurance. There is also, again in my humble opinion, not enough emphasis on professional responsibility and accountability.

OFFUTT: Although testing is not the only topic that will help develop high-quality software, it is a reasonable measure of how universities value reliability and quality in software. Most CS departments don’t even offer an undergraduate testing course. Very few CS degrees require a software testing course. Testing is usually taught as a two-week topic in a general software engineering overview course. A time-to-market

Most commercial products do not need to meet the requirement of high assurance; time-to-market instead often has the priority.
priority is sometimes the right business priority, although it’s probably the best priority less often than managers think. How much testing is needed in a particular project depends on the software and is actually quite difficult to assess. Although we have some excellent studies that show that the cost of debugging and fixing failures after deployment is significantly more than the cost before deployment (G. Tassey, editor, The Economic Impacts of Inadequate Infrastructure for Software Testing, NIST tech. report 7007.011, May 2002), it varies quite a bit. Most universities don’t teach this kind of software economics analysis.

LEWIS: There needs to be a much greater emphasis placed on software quality in the undergraduate curriculum. When I think of some of the students I have known, the thought of a computer-controlled throttle in my car is terrifying. It’s partly the nature of the beast. Students are frequently struggling just to get something that works. Projects are often of a nature so that students don’t have to live with and correct the consequences of bugs or poor design. Also, the emphasis on high-assurance software needs to be made throughout the curriculum, so that students don’t pick up bad habits, but faculty are not always aware of what is considered best practice in software development.

LAPLANTE: I think every science and engineering program should have some kind of systems thinking course in the curriculum that addresses these kinds of problems. If not, then those programs are doing their students and society a disservice. Most complex and even noncomplex systems are or can be wirelessly connected, creating unforeseen interactions of noncritical systems with critical systems. And we already discussed the need for a mathematical understanding of system composability, interoperability, testability, and so on. There are privacy risks as well, and these should be taught in every engineering and science program.

ROKNES: This is an important question that raises concerns for governments, universities, commerce, and individuals. CS education focuses on efficient algorithms and implementations and generally does not provide courses on reliability and safety. Part of the reason for this is that there is much to be taught and learned when taking up fundamental CS concepts and programming techniques. Nowadays, with the proliferation of new products with embedded computing capabilities and the trend toward connecting legacy devices to the network, there is a definite need to include curriculum materials that emphasize reliability and high-assurance software. The questions arise as to what can be omitted in the classical CS curriculum and who will be able to teach the new curriculum because few graduate programs are researching questions related to reliability and safety.

COMPUTER: Is it time to establish software engineering departments, separate from CS, to accommodate industrial needs for software engineers? Some universities have already begun this process. Do you see it accelerating?

OFFUTT: Yes! To me it’s inevitable. CS is fissioning into computing fields such as software engineering, information systems, security, and so on. I don’t think the structure of computing has fully emerged yet, but I am 100 percent certain that in 50 years, CS (and math) will be at the core of several related but distinct computing fields, exactly as physics (and math) is currently at the core of several related but distinct engineering fields.

MILLER: I think it is accelerating, and I think it is a good development. This is not a new idea. Way back in 1999, David Parnas wrote a well-cited paper called “Software Engineering Programs are Not Computer Science Programs” (IEEE Software, vol. 16, no. 6, 1999, pp. 19–30).

SHIEH: Software engineering is considered one of the core disciplines in CS. CS has grown quickly, so many new areas have evolved in recent years. If the software industry maintains its current fast-growing trend, it won’t be a surprise to see many software engineering departments established to meet the market demand.

LAPLANTE: I think it is time, but I do not know of many universities that have taken or are taking this step. I don’t think it is going to happen at a fast pace because there would be lots of internal, university politics that get in the way, as well as the challenges involved in creating a new department. This is where I’d like to see some big corporations step up and fund such departments.

LEWIS: I’m not convinced a separate department of software engineering is required for pedagogical reasons. The IEEE Software Engineering Body of Knowledge, for instance, looks like a CS degree with elective that reflects a specialization in software engineering. The trend of having separate departments of software engineering is much more common outside the US. In the US, the more common practice seems to be for CS departments to offer undergraduate and MS degrees or concentrations in software engineering.
ROKNE: There are many computer science and software engineering departments going by a great variety of names: “Electrical and Computer Engineering,” “Electrical, Computer and Software Engineering,” “Software Engineering,” “Computer Science” are some of the more common ones. What I see is new variations of department names and expertise rather than many departments named “Software Engineering.”

COMPUTER: Past CS departments prided themselves on teaching more theory (for example, Shannon’s information theory) than practice. Is this still true?

ROKNE: I do not think this is possible for this to still be true. Most CS departments now aim to teach a reasonable mix of courses that have foundational content (theoretical knowledge) combined with courses that have current relevance containing applied techniques and programming tools.

LAPLANTE: I think that less theory is taught now than 20 years ago. I do lament the absence of basic AI courses, automata theory, risk, analysis, and decision making from most CS curricula. These types of courses help CS graduates really understand how computing can (and cannot) improve the human situation. And it’s when students learn theory that they can push back against ill-informed accusations blaming computers and software for system failures (when it is more often the human element, bad policy, hardware, or the environment).

OFFUTT: Why do people study CS to become software engineers? Do students study physics to become civil engineers? No, they study lots of physics and math for two years, and spend most of the next two years studying engineering. What does the software industry and the society at large need? We need people very knowledgeable of the theory in computer science, just like we need truly knowledgeable physicists. But we need orders of magnitude more civil, mechanical, geotechnical, and aerospace engineers who have a general knowledge of theory but a deep knowledge of how to apply physics (and math) to create usable, safe, maintainable, reliable, and secure software applications [J. Offutt, “Putting the Engineering into Software Engineering Education,” IEEE Software, vol. 30, no. 1, 2013, pp. 96–100].

LEWIS: I’d allow that CS departments still do place a premium on theory in their curricula, particularly graduate research departments. A cynic might say it’s because teaching theory means

ROUNDTABLE PANELISTS

Phillip A. Laplante is a professor of software and systems engineering at Pennsylvania State University. His research interests include software project management, software requirements engineering, and the Internet of Things. Laplante received a PhD in computer science from Stevens Institute of Technology. Contact him at plaplante@psu.edu.

Michael Lewis is an associate professor and chair of the Department of Computer Science at the College of William and Mary. His research interests include algorithms for nonlinear optimization and engineering and scientific computation. He received a PhD in mathematical sciences from Rice University. Contact him at rmlewi@wm.edu.

Keith Miller is the Orthwein Endowed Professor for Lifelong Learning in the Sciences at the University of Missouri–St. Louis. His research interests include computer ethics, software testing, and online learning. Miller received a PhD in computer science from the University of Iowa. Contact him at milerkei@umsl.edu.

Jeff Offutt is a professor of software engineering at George Mason University. His research interests include software testing, web software engineering, and software engineering education Offutt received a PhD in computer science from the Georgia Institute of Technology. Contact him at offutt@gmu.edu.

Jon George Rokne is a professor of computer science at the University of Calgary. His research interests include numerical analysis, computer graphics, and social networks. Rokne received a PhD in mathematics from the University of Calgary. Contact him at rokne@ucalgary.ca.

Shihpyng Shieh is a University Chair Professor at National Chao Tung University. His research interests include intrusion detection and network and system security. Shieh received a PhD in electrical and computer engineering from the University of Maryland, College Park. Contact him at ssp@cs.nctu.edu.tw.
you don’t have to revise your lecture notes very often (if ever). But theory helps us differentiate CS from programming. There are many excellent programmers and computer professionals without formal training in CS. But the theoretical component of the CS curriculum helps train students to think abstractly when trying to solve practical problems, which is why we value it so much. Theory also constitutes the eternal verities in CS, the fixed stars in a field that is otherwise constantly changing. What I learned about algorithmic complexity is still pretty useful; what I learned about APL, not so much.

**MILLER:** I think that emphasis is institution and professor specific. Some programs work hard to make sure that their graduates know at least a modicum of theory. Other programs do not. I would speculate that most undergraduate programs (especially those not connected tightly to a mathematics department, or without graduate programs) now emphasize practice more and theory less. ACM and IEEE Computer Society have long differentiated CS from other types of curriculum, including computer engineering, information systems, information technology, and software engineering. Each of these subdisciplines will have its own set of theories, and its own amount of emphasis on those theories and on practice.

**SHIEH:** Theory and practice are equally important. Theory has more influence if it can be used in practice. Similarly, techniques in practice can be more powerful if they are based on theoretical analysis. Although some techniques are very useful in dealing with real problems in practice, theory is still the key to advancing technologies.

Although technology has evolved significantly over the last 40 years, the question remains as to whether CS curricula have kept up with the changes. The panelists agreed that some trends are more widely recognized by educational institutions than others, such as the growing need for more experienced practitioners. However, their opinions varied on the need for separate software engineering departments and teaching methods for reliability, third-party components, security, and privacy.

Clearly, to produce professionals who are prepared for the ever-changing real world, educational institutions must strategically address ever-changing IT trends. This is a new age of CS education—no going back.

---

**JEFFREY VOAS** is a computer scientist at the National Institute of Standards and Technology (NIST). Contact him at j.voas@ieee.org.

**RICK KUHN** is a computer scientist at NIST. Contact him at d.kuhn@nist.gov.

**CELIA PAULSEN** is a researcher at NIST. Contact her at celia.paulsen@nist.gov.

**KIM SCHAFFER** is a researcher at NIST. Contact her at kim.schaffer@nist.gov.