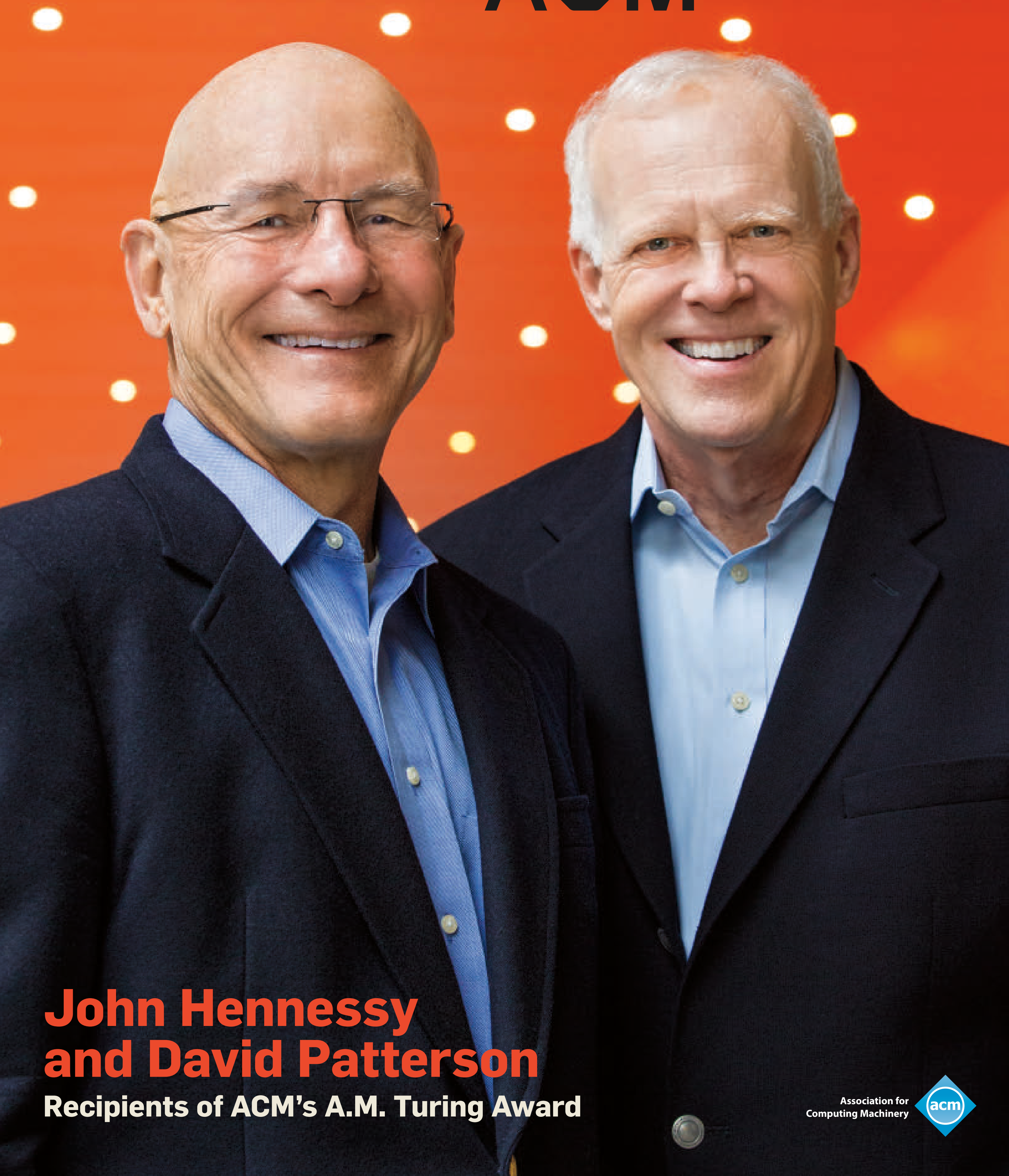
COMMUNICATIONS OF THE ACM

CACM.ACM.ORG

06/2018 VOL.61 NO.06



John Hennessy and David Patterson

Recipients of ACM's A.M. Turing Award

Association for Computing Machinery

SIGCOMM 2018

BUDAPEST
AUGUST 20-25

LEARN THE FUTURE OF DATA COMMUNICATION

Mentoring
One-on-one meetings
Main conference
Industrial demos
Posters and demos
Student research competition
Tutorials
Workshops
Travel grants
Topic preview
Hackathon

conferences.sigcomm.org/sigcomm/2018

GENERAL CHAIRS
Sergey Gorinsky / IMDEA, Spain
János Tapolcai / BME, Hungary

TREASURER
Tilman Wolf / UMass, USA

PROGRAM COMMITTEE CHAIRS
Mark Handley / UCL, UK
Dina Katabi / MIT, USA

WORKSHOP CHAIRS
Xiaoming Fu / GAU, Germany
K. K. Ramakrishnan / UCR, USA

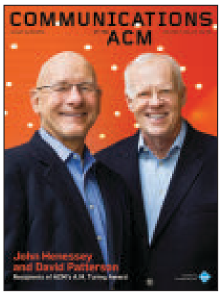
acm Association for Computing Machinery

acm sigcomm

Association for Computing Machinery
STEM
Author's Rights
SIGs
Science
Open
Copyright
Machine Learning
Emerging Technologies
CACM
Digital Library
Articles
Research
Transactions
OpenSurround Service
Gold Open Access
TechNews
Publishing
Magazines
Cybersecurity
Proceedings
Articles
eLearn
IT
3D Printing
Community
Big Data
Robotics
Cloud
Author-Izer
CHORUS
Networks
HCI
OpenTOC
Journals
IoT
Access
HPC
XRDS
AI
Ubiquity
Technology
Conferences
Computing
Green Open Access
ACM
Software
Hybrid Open Access
Communications
Computer Science
Scholarly
Inroads
Computers in Entertainment
Queue

COMMUNICATIONS OF THE ACM

06/2018
VOL. 61 NO. 06



About the Cover:
David A. Patterson (left) and John L. Hennessy, recipients of the 2017 ACM A.M. Turing Award, photographed on the campus of Stanford University last March. Photographed by Richard Morgenstein; http://www.morgenstein.com

Association for Computing Machinery
Advancing Computing as a Science & Profession

COMMUNICATIONS OF THE ACM

Trusted insights for computing's leading professionals.

Communications of the ACM is the leading monthly print and online magazine for the computing and information technology fields. *Communications* is recognized as the most trusted and knowledgeable source of industry information for today's computing professional. *Communications* brings its readership in-depth coverage of emerging areas of computer science, new trends in information technology, and practical applications. Industry leaders use *Communications* as a platform to present and debate various technology implications, public policies, engineering challenges, and market trends. The prestige and unmatched reputation that *Communications of the ACM* enjoys today is built upon a 50-year commitment to high-quality editorial content and a steadfast dedication to advancing the arts, sciences, and applications of information technology.

ACM, the world's largest educational and scientific computing society, delivers resources that advance computing as a science and profession. ACM provides the computing field's premier Digital Library and serves its members and the computing profession with leading-edge publications, conferences, and career resources.

Subscriptions
An annual subscription cost is included in ACM member dues of $99 ($40 of which is allocated to a subscription to *Communications*); for students, cost is included in $42 dues ($20 of which is allocated to a *Communications* subscription). A nonmember annual subscription is $269.

ACM Media Advertising Policy
Communications of the ACM and other ACM Media publications accept advertising in both print and electronic formats. All advertising in ACM Media publications is at the discretion of ACM and is intended to provide financial support for the various activities and services for ACM members. Current advertising rates can be found by visiting http://www.acm-media.org or by contacting ACM Media Sales at (212) 626-0686.

Single Copies
Single copies of *Communications of the ACM* are available for purchase. Please contact acmhelp@acm.org.

COMMUNICATIONS OF THE ACM
(ISSN 0001-0782) is published monthly by ACM Media, 2 Penn Plaza, Suite 701, New York, NY 10121-0701. Periodicals postage paid at New York, NY 10001, and other mailing offices.

POSTMASTER
Please send address changes to
Communications of the ACM
2 Penn Plaza, Suite 701
New York, NY 10121-0701 USA

Printed in the USA.

Association for Computing Machinery

PLEASE RECYCLE THIS MAGAZINE

DOI:10.1145/3210376

Vinton G. Cerf

Celebrating Excellence

As we do every year, ACM convenes a gala event to celebrate and honor colleagues in our computing universe who have achieved pinnacle success in the field.

Our most prestigious recognition is the ACM A.M. Turing Award and the 2017 award goes to John Hennessy and David Patterson:

"For pioneering a systematic, quantitative approach to the design and evaluation of computer architectures with enduring impact on the microprocessor industry."

Their primary insight was to find a method to systematically and quantitatively evaluate machine instructions for their utility and to eliminate the least used of them, replacing them with sequences of simpler instructions with faster execution times requiring lower power. In the end, their designs resulted in Reduced Instruction Set Complexity or RISC. Today, most chips make use of this form of instruction set. A complete summary of their accomplishments can be found within this issue and at the ACM Awards website.[a]

The second most significant award from ACM is the ACM Prize in Computing and the 2017 award goes to Dina Katabi of the Massachusetts Institute of Technology's Computer Science and Artificial Intelligence Laboratory (MIT CSAIL). Her contributions to wireless communication and applications are both deep and broad. Concerned about interference in wireless networks, Katabi developed the concept of *network coding* in which messages are encoded and recoded as they move through network in such a way that increased resilience and capacity can be achieved. She also pioneered creative work on the use of Wi-Fi signals to perceive objects and life forms on the other side of walls opaque to visual frequencies. The ability to analyze these extremely weak signals seems truly straight out of science fiction! A summary of her work also can be found at the ACM Awards website.[b]

A third recognition—the 2018–2019 Athena Lecturer Award—goes to Andrea Goldsmith of Stanford University. In addition to her contributions to the theory and practice of adaptive communications, she is also honored for her successful transfer of technology into practice. Her theories of time-varying adaptive modulation led to significant improvements in EDGE, Wi-Fi and 3GPP/LTE. She established a company—Quantenna—that developed the first adaptive Multi-in/Multi-out 4X4 antenna for 802.11N Wi-Fi applications. For more information see the ACM Awards website.[c]

There are many more awards to be given and people to be recognized for their extraordinary accomplishments on June 23 in San Francisco. We take great pride in knowing that our colleagues continue to make significant and constructive contributions to our society. I hope that many of you will be able to join us for the celebration of their work.

Speaking of celebrations, it is fitting and timely to recognize the work of our senior leadership whose terms come to an end as these celebrations conclude. Graduating to Past President, Vicki Hanson will complete her successful two-year term as President of ACM at the end of June. Cherri Pancake will finish her term as Vice President and Elizabeth Churchill her term as Secretary-Treasurer. We owe much to all three of these dedicated women whose careers and experience have placed them in leadership roles in ACM. I have no doubt that they will continue to contribute to ACM in a variety of ways, as they have in the past. All three are important and much-needed role models for all who work in computer science.

Artificial intelligence and, especially, machine learning, are experiencing another form of celebration as success after success is reported in diverse applications ranging from board gaming to real-time control of cloud computing cooling systems to self-driving cars. While we can take pride in the constructive aspects of these new tools of computer science, we must also be alert to the quirks, idiosyncrasies, and weaknesses of these emerging technologies. As responsible members of our profession, it is important to help policymakers and the general public to adjust their expectations of these techniques to ensure they are applied with due consideration for their limitations.

I hope to see many of you at the ACM Awards Banquet! C

Vinton G. Cerf is vice president and Chief Internet Evangelist at Google. He served as ACM president from 2012–2014.

a https://amturing.acm.org/

b https://awards.acm.org/acm-prize

c https://awards.acm.org/athena

DOI:10.1145/3210585

When to Hold 'Em

Neil Savage deserves praise for his informative overview of recent computational results related to Nash equilibrium in his news story "Always Out of Balance" (Apr. 2018). I fully agree that the notion of Nash equilibrium does not always reflect how competitors behave in competitive situations, and that the fact that Nash equilibrium is provably computationally intractable makes it less useful than John Nash himself might have envisioned when he developed it. However, Savage also overstated (somewhat) the effect of intractability by claiming the intractability of computing Nash equilibrium necessitates researchers abandon this notion in favor of other competition-related ideas.

While looking for Nash equilibrium yields additional computational complexity, the decision-making problem is, in general, already computationally intractable (NP-hard) for non-competitive situations (such as when a company makes internal planning decisions). In doing so, a company would be looking for an optimal solution (such as one that would aim to help produce maximum profit), but computational optimization is, in general, NP-hard. Such computational intractability does not mean researchers have to abandon the idea of optimization and look for other ideas. Many real-life problems are NP-hard (such as robotic movement) and what makes working on them such an intellectual and computational challenge.

Indeed, there is no general feasible algorithm (unless P = NP), so computer scientists need to be creative when designing algorithms for specific practical problems.

Vladik Kreinovich, El Paso, TX, USA

Beware the Internet of Traitorous Things

Stephen B. Wicker's Viewpoint "Smartphones, Contents of the Mind, and the Fifth Amendment" (Apr. 2018) correctly pointed out the importance of protecting against the risk of self-incrimination through our smartphones but neglected the far-greater risk the more pervasive Internet of Things (IoT) poses to our personal liberty and the future of global e-commerce.

IoT can be traitorous in two ways: to their users by acting against their interests by exploiting sensitive personal information and to freedom of global e-commerce by motivating nations to prohibit the use of IoT connected to foreign-domiciled data centers due to threats of foreign mass surveillance.

In particular, future holographic-type glasses, or "hologlasses," could be used against their users in ways more insidious than smartphones are today, as discussed by Wicker. Hologlasses promise all the functionality of today's smartphones—but without the disadvantage of having to look down while holding the device. The world's largest computer companies have publicly discussed their expectation that hologlasses could be in use at the scale of today's smartphones by 2030.

Moreover, hologlasses promise many applications that are inherently impractical through smartphones alone. For example, they likely will include facial recognition to instantly tag people a user might encounter, providing real-time relevant background information, including even ongoing analysis of their emotional states. Some police officers in China are already using early-model hologlasses with facial recognition to nab those in a police database as they walk by.

A backdoor in hologlasses could enable a "we see and hear what you see, hear, and do" capability to provide extraordinary insight into a user's private thoughts. Protection against self-incrimination through hologlasses could become a contentious legal (and political) issue, way beyond Wicker's discussion of smartphones.

Beyond threats of personal self-incrimination, defending against the threat of foreign mass surveillance could also make freedom of global commerce in IoT problematical. Sovereign nations could end foreign mass surveillance of their own citizens and institutions by prohibiting control of their IoT devices, including hologlasses, by companies domiciled in other nations. China has already adopted a strategy of having China-domiciled companies take over large information-based businesses that operate in China.

Storing sensitive information in data centers of foreign-domiciled companies could wind up being widely banned if China's strategy for ending foreign mass surveillance within its sovereign borders is adopted by other countries. Outside of China, international companies doing business online might still hope to preserve their advertising-based business models by continuing to function as brokers, with the big difference being that matchmaking between merchants and users would be done on the users' own equipment as opposed to in the data centers of foreign-domiciled companies.[2] A new policy of storing sensitive information on users' own equipment could address increasing European concerns about ending foreign mass surveillance thereby conforming to recent judgments of the European Court of Justice.[1]

References

1. European Court of Justice. *Judgment of the Court*, Dec. 21, 2016; https://publications.europa.eu/en/publication-detail/-/publication/a442d1ac-f6fd-11e6-8a35-01aa75ed71a1/language-en
2. Hewitt, C. Islets protect sensitive IoT information: Verifiably ending use of sensitive IoT information for mass surveillance can foster (international) commerce and law enforcement. *Social Science Research Network*; https://papers.ssrn.com/sol3/papers.cfm?abstract_id=2836282

Carl Hewitt, Palo Alto, CA, USA

Author Responds:

I agree. Hologlasses do raise the prospect of third parties capturing everything we perceive and our responses to those perceptions. Such ultimately personal information would support near-ideal modeling of the individual and finely tuned manipulation that goes well beyond our current concerns regarding electoral interference and Facebook advertising. We can only hope the U.S. Supreme Court will recognize the increasing anachronism and folly of the third-party doctrine, and that Congress finds its way to meaningful limitations on data collection. We can also hope that some sense of personal aesthetics will keep our future selves from wearing hologlasses for all but the briefest periods of time.

Stephen B. Wicker, Ithaca, NY, USA

Called to Programming

I wonder what could have concerned David G. Stork so much about the equity of women in technology that he was compelled to write a letter to the editor, "Gender 'Equity' in Computer Science" (Apr. 2018), on Jodi L. Tims's "From the Chair of ACM-W column "Achieving Gender Equality: ACM Can't Do It Alone" (Feb. 2018).

When I first began working as an engineer after college, I was disappointed to find I basically worked only with men, usually older and married. I learned programming largely on my own since I was eight years old by retyping simple programs out of magazines like *Compute!'s Gazette* and experimenting by changing the code or partially running the code as I typed it. While I did learn programming more professionally in college, it was something more within me that drew me to the field.

No one not already interested in technology can be forced to care about programming. It is certainly interesting to some people, but to many others it is just a tool, like, say, a screwdriver. People can know all about building screwdrivers, all the different sizes and types of heads, materials, and qualities, but all one really needs to know is that it is useful for tightening screws that are usually useful for something far more important than the screwdriver itself. Following the screwdriver analogy, it would not be that different to ask why we don't see more women mechanics at the local garage. Yes, I know there are some; in fact, one changed my car's oil the last time it needed it.

People ultimately do what they want to do, not what education (or even well-meaning parents) directs them to do.

Robert Wilkens, Levittown, NY, USA

Communications welcomes your opinion. To submit a Letter to the Editor, please limit yourself to 500 words or less, and send to letters@cacm.acm.org.

BLOG@CACM

The *Communications* Web site, http://cacm.acm.org, features more than a dozen bloggers in the BLOG@CACM community. In each issue of *Communications*, we'll publish selected posts or excerpts.

twitter
Follow us on Twitter at http://twitter.com/blogCACM

DOI:10.1145/3204443 http://cacm.acm.org/blogs/blog-cacm

Programming Programming Languages, and Analyzing Facebook's Failure

Mark Guzdial considers an idea with significant educational implications, while Susan Landau looks into the Cambridge Analytica/Facebook scandal.

Mark Guzdial
A Computing Education Research Perspective on Programmable Programming Languages
http://bit.ly/2JcSr4T
March 8, 2018

When the March *Communications* appeared in my mailbox, my monthly Blog@CACM post wrote itself. The cover story, an important idea with significant education implications, was *A Programmable Programming Language* by Matthias Felleisen, Robert Bruce Findler, Matthew Flatt, Shriram Krishnamurthi, Eli Barzilay, Jay McCarthy, and Sam Tobin-Hochstadt. The authors describe their work with Racket, where software developers redesign the programming language to match a problem being solved.

In the ideal world, software developers would analyze each problem in the language of its domain, then articulate solutions in matching terms. They could communicate with domain experts and separate problem-specific ideas from the details of general-purpose languages and specific program design decisions.

By "language," we mean a new syntax, a static semantics, and a dynamic semantics that usually maps the new syntax to elements of the host language and possibly external languages via a foreign-function interface (FFI).

The article does a good job of playing out the issues and possibilities. I'm particularly taken with the explanation of how types play an important role in specifying the new languages. The authors have thought a lot about how a language can usefully constrain and facilitate programmers' work to improve problem-solving and productivity.

To fit into *CACM*, the paper is necessarily short, so not all the issues are laid out. I know from discussions with Krishnamurthi these ideas *started* from their work in education with DrRacket. The team wanted to define subsets of the language that were easier to teach, and generalizing that idea led to this work.

Programmable programming languages have been created in the past. Smalltalk-72 objects could re-parse their input; the language included an "eyeball" character in methods that could look ahead in the token stream to parse the rest of the input line. Dan Ingalls wrote in the "Smalltalk-80: Bits of History, Words of Advice" that this made Smalltalk-72 difficult to grow. The programmer was challenged to design a new language that could be understandable by others. How do programmers afterwards know the context and thinking of the programmer who implemented this language? How do they learn "Smalltalk-72" when it's really an umbrella for a bunch of different languages? This modern work is richer for thinking about developing new programming languages for particular problems, but the educational issues are still there.

Programming is not just about problem-solving.

▸ *Programming is a job skill.* It is hard for students to learn programming. Students often learn a particular programming language to get themselves jobs. Are problem-specific programming languages easier to learn? Will some become so popular and useful that it's worthwhile (from a job perspective) for a student to learn a problem-specific programming language?

▸ *Programming creates infrastructure.* Programming is the infrastructure for our world. There are large systems still in use today written in Cobol and PL/1, and we have to maintain that information infrastructure. Does creating more languages in problem-specific forms make it harder to find enough programmers who know infrastructure built in these problem-specific forms? Is it harder or easier to train new programmers to maintain that infrastructure?

▸ *Intellectual property is embodied in programming.* Programs are intellectual property. When intellectual property is defined in terms of thousands of lines of code, IP can't be easily carried away in someone's head. But when the IP is in the *language*, which can be learned and carried in a single person's head, the definition of what's protected and what was just *learned* seems complicated. If a programmer moves from Company A to Company B, and that programmer has learned the problem-specific language in Company A, and then re-implements it in Company B, was intellectual property stolen?

I love that this article makes all of software soft again; everything is malleable, down to the programming language itself. As an education researcher, I know learning programming is a struggle. New problem-specific languages may increase the challenges for computing education research. Because of the value of these kinds of languages, the new research questions raised are worth investigating.

Susan Landau
What Went Wrong? Facebook and 'Sharing' Data with Cambridge Analytica

http://bit.ly/2uFPjv3

March 28, 2018

The road to the Cambridge Analytica/Facebook scandal is strewn with failures. There's the failure to protect users' privacy, the failure to protect voters, and the failure to uncover the actions and violations of laws that may well have affected the Brexit referendum and the U.S. presidential election. The latter two threaten the heart of democracy.

I want to focus on the failure to protect users' privacy, which has the virtue of being easy to unpack, even if its resolution is far from simple. This failure to protect privacy has multiple parts.

First, there's Facebook's failure to protect user privacy. Indeed, the company's aim was quite the opposite. Facebook believed "every app could be social." That meant giving broad access not only to a user's data, but also to that of his "friends." In 2013, Cambridge University researcher Aleksandr Kogan paid 270,000 Facebook users to take a personality quiz (the money came from Cambridge Analytica, but that's another part of the story). Doing so gave Kogan's app the ability to "scrape" information from their profiles. In those days, Facebook's platform permitted the app not only to access the quiz takers' profiles and "scrape" information from them; the social network also allowed apps to do the same to the profiles of the quiz takers' "friends"—50 million of them.

Data from the friends would be collected unless the friends explicitly prohibited such collection. The ability to do so was not easy; users were not told their data would be shared if a Facebook friend engaged an app that did such scraping. To prevent collection, users had to first find out that their friends' apps would do this, then configure their profiles to prevent such data sharing.

Then there's Facebook's failure to take legal action after the company became aware the data of those 50 million Facebook users had been provided to Cambridge Analytica. This data transference violated Kogan's agreement with Facebook in acquiring the data. When Facebook found out, it requested Cambridge Analytica certify they had destroyed the user files; the Silicon Valley company did not ensure Cambridge Analytica had done so. As we know, Cambridge Analytica did not comply. Facebook's failure to ensure the files were destroyed was failure number 2.

Finally, there's Facebook's failure to inform the 50 million users whose data was taken. There was a breach of contract between Kogan and Facebook, but there was also a privacy breach: the profiles of 50 million Facebook users were being used by Cambridge Analytica, a British firm specializing in using personal data for highly targeted, highly personalized political ads. Facebook failed to inform those 50 million users of the breach. That was failure number 3.

Facebook is on the way to repairing some of these failures. In 2014, Facebook limited the information apps could collect on Facebook friends—though not fully. Mark Zuckerberg said Facebook will, belatedly, inform the 50 million Facebook users of the privacy breach that happened in 2014.

But there are other failures as well.

The fourth is society's, which hasn't been taking privacy particularly seriously. This isn't universally true. In the U.S., for example, the Federal Trade Commission (FTC) and the states' Attorneys General have taken companies to court when the firms fail to protect users' privacy or fail to follow their own privacy policies. But their set of tools for doing so is quite limited. There are a handful of laws that protect privacy in particular sectors. There are fines if companies fail to live up to stated privacy policies. There are data breach laws. And there's the ability to fine if actual harm has been caused.

The Facebook/Cambridge Analytica case is garnering significant attention by both the FTC and the states' Attorneys General. It helps that in 2011, the FTC acquired a powerful tool when Facebook signed a consent decree as a result of an earlier privacy violation, which required the company to make clear "the extent to which [Facebook] makes or has made covered information accessible to third parties." Did the fact that those 50 million "friends" had difficulty preventing collection of their data constitute a violation of the consent decree? We will find out. At a potential $40,000 per violation, the consequence could be quite expensive for Facebook.

There's a fifth failure that may well be most important of all: our willingness to trade data about ourselves for seemingly free services. That's nonsense; services that manipulate how you spend your time, how you feel, and whom you vote for are anything but free. Maybe it's time to cut that connection? Some things will be harder to lose—seeing that photo of your nephews clowning at breakfast, or getting updates from the folks at your previous job—but you may find an extra hour in your day. That's an hour to call a friend, take a walk, or read a book. It sounds like a good trade to me. I wouldn't actually know; I value my privacy too much, and never joined the network in the first place.

Mark Guzdial is a professor in the Computer Science & Engineering Division, and jointly in the Engineering Education Research program, of the University of Michigan. Guest blogger **Susan Landau** is a computer scientist and cybersecurity policy expert at the Fletcher School of Law & Diplomacy and the School of Engineering, Department of Computer Science, Tufts University.

news

Profile | DOI:10.1145/3204451

Neil Savage

Rewarded for RISC

ACM A.M. Turing Award recipients David Patterson and John Hennessy developed the "dangerous" idea that software should be simpler so it can be executed more quickly, which evolved into the Reduced Instruction Set Computer architecture.

It was the early 1980s, and microprocessors were making the transition from laboratory curiosity to commercial product. To make them work, many computer scientists were trying to copy the same complex instructions used in mainframe computers. In fact, some wanted to expand those instructions, trying to get the buggy software of the era to work better.

However, two young professors had a different suggestion. "John and I come along and say absolutely the opposite. Not only should we not make it more complicated, we should make it even simpler," says David Patterson, who at the time was an assistant professor of computer science at the University of California, Berkeley. "We weren't just criticizing the trend. We were making an argument that people thought was dangerous, and was just going to make software fail more."

That "dangerous" idea promoted by Patterson and John Hennessy, then an assistant professor of computer science at Stanford University, was the Reduced Instruction Set Computer (RISC) architecture, which relied on a simpler collection of general functions the processor would perform, shrinking the number of transistors needed to carry out a task. Today, 99% of the more than 16 billion microprocessors produced each year are based on RISC architecture, powering smartphones, tablets, and the Internet of Things, and earning Hennessy and Patterson this year's 2017 ACM A.M. Turing Award.

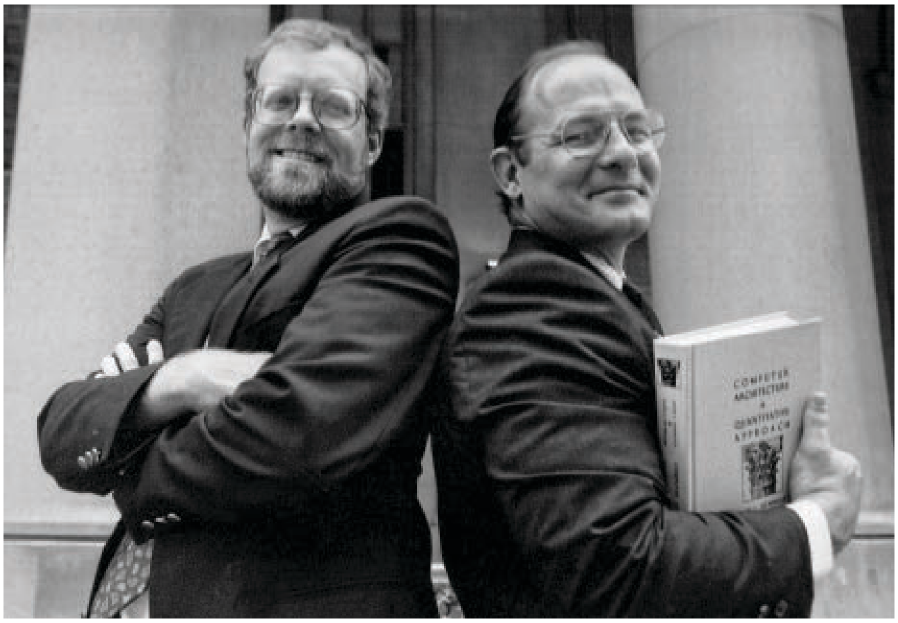
It wasn't that the number of instructions was necessarily smaller, Hennessy says; it was that they were less complex. "We said it's not how big the program is, it's how fast the program runs," he says. Those early machines had perhaps 25% more instructions under RISC design, but ran them five times as fast. The advantage of the approach was so great, Patterson says, that processors designed that way could outperform those from talented designers using older methods. "The Berkeley RISC and Stanford MIPS processors designed by grad students were probably better microprocessors than what Intel could build," he says. In fact, Patterson's first prototype, which had 44,000 transistors, outperformed a 100,000-transistor device made the conventional way.

"We said it's not how big the program is, it's how fast the program runs," Hennessy recalls.

MIPS—Microprocessor without Interlocked Pipeline Stages—was Hennessy's RISC architecture, which sped up processing by using instructions loaded from the memory into a register, which could be accessed faster. Hennessy founded a semiconductor design company, MIPS Computer Systems, in 1984 to commercialize the technology, spurred in part by doubts from the computer industry that the approach would work in the real world. Around the same time, Bill Joy of Sun Microsystems in Santa Clara, CA, became interested in RISC, and brought Patterson on as a consultant.

Both projects were successful, which won over the skeptics. "When people start making money, it's hard to argue

PHOTO BY RICHARD MORGENSTEIN PHOTOGRAPHY



that it doesn't work," Patterson says.

The men, of course, were academics as well as processor designers, and they were unsatisfied with how computer architecture was taught. "We were really unhappy with the quality of textbooks," says Hennessy. "They were descriptive and comparative, but not in a numeric and quantitative fashion." So they developed ways to measure whether a new processor design was an improvement, using metrics such as cost and speed. In 1989, they published their textbook, *Computer Architecture: A Quantitative Approach*. A year later, they published a version for undergraduates. Both are now in their sixth edition.

Garth Gibson, a professor of computer science at Carnegie Mellon University in Pittsburgh, PA, was a student in Patterson's lab in the mid-1980s. "The book became a very good way for a lot of people to understand how computer architecture works and do a very good job at it," Gibson says.

RISC has evolved as well, though many of the basics remain the same. The current version is RISC-V, an open source instruction set architecture allowing designers to add extensions that optimize a particular application. Such domain-specific architectures are the only option for improving performance as the benefits of Moore's Law start to fade. "The circuits aren't getting faster any more, so you're going to have to change the instruction set architecture," Patterson says with an air of excitement.

The impending end of Moore's Law, and the demise of Dennard scaling, which said that as transistors get smaller their power density stays constant, are imposing new limitations on architecture, Hennessy says, and that makes RISC even more important. "Before, when transistors were doubling every two years or so, if you waste a few transistors, who cares? But when they're not going up that fast, then efficiency becomes very important," he says.

Patterson officially retired from Berkeley in 2016, although he still works there part-time. He also works at Google, developing domain-specific architectures for machine learning. Hennessy was president of Stanford from 2000 to 2016, and now directs the Knight-Hennessy Scholars program, which provides full scholarships to Stanford to graduate students from around the world. The program has just accepted its first class of 49 students from 16 countries.

Hennessy says the quantitative and engineering approach he brought to computer architecture served him well as president when the financial crisis hit in 2008 and Stanford, like other schools, lost a quarter of its endowment. He decided the university had to cut its budget and do layoffs in one fell swoop, rather than parceling out the pain over time as many other institutions did. The next year its finances had stabilized, and Stanford was able to hire faculty and recruit new graduate students again. "We had one harrowing year, and then things were better," he says.

The men will share the Turing Award's $1-million prize, supported by Google. Hennessy, who says he's been fortunate in life, plans to donate his half to charity. Patterson, with several children and grandchildren who will need to pay college tuition, says he'll invest his in education as a consumer.

The two came to computing along different paths. Patterson stumbled into it; during his senior year as a math major at the University of California, Los Angeles, a math class he meant to take was cancelled, so instead he took a course in programming with Fortran. The experience grabbed him, and he went on to earn an M.S., then a Ph.D., from UCLA in computer science. He applied for a job at Berkeley and, when he had not heard a response, his wife urged him to call and ask, which got him an interview and, eventually, a job.

Hennessy was interested in computers in high school, and as a science project built a machine that used relays to play tic-tac-toe. "It both won me a prize and it also helped me win my wife, because it impressed her family sufficiently that they thought 'well, maybe this guy's going to be okay'," he says. He earned a B.S. in electrical engineering from Villanova University, and an M.S. and Ph.D. in computer science from the State University of New York at Stony Brook.

Both encourage young computer scientists to take risks as they did. "It wasn't clear that that was the safest path to tenure, to rock conventional wisdom, but we believed in what we were doing, and that worked out pretty well," Patterson says.

Hennessy agrees, "You have to be a little fearless, willing to take some chances and work on things that are a little contrarian." C

Neil Savage is a science and technology writer based in Lowell, MA.

Watch Patterson and Hennessy discuss their work in this exclusive *Communications* video. https://cacm.acm.org/videos/2017-acm-turing-award

Science | DOI:10.1145/3204445

Chris Edwards

Deep Learning Hunts for Signals Among the Noise

Neural networks can deliver surprising, and sometimes unwanted, results.

Over the past decade, advances in deep learning have transformed the fortunes of the artificial intelligence (AI) community. The neural network approach that researchers had largely written off by the end of the 1990s now seems likely to become the most widespread technology in machine learning. However, protagonists find it difficult to explain why deep learning often works well, but is prone to seemingly bizarre failures.

The success of deep learning came with rapid improvements in computational power that came through the development of highly parallelized microprocessors and the discovery of ways to train networks with enormous numbers of virtual neurons assembled into tens of linked layers. Before these advances, neural networks were limited to simple structures that were easily outclassed in image and audio classification tasks by other machine-learning architectures such as support vector machines.

Theorists have long assumed networks with hundreds of thousands of neurons and orders of magnitude more individually weighted connections between them should suffer from a fundamental problem: overparameterization. There are so many weights that determine how much each neuron influences its neighbors that the network could simply find a way to encode the data used to train it. It would then correctly classify anything in the training set, but fail miserably when presented with new data.

In practice, deep neural networks do not fall easily into overparameterization; instead, they are surprisingly good at dealing with new data. When trained, they seem able to ignore parts of images used for training that had little bearing on classification performance, rather than trying to build synaptic connections to deal with them.

Stefano Soatto, professor of computer science at the University of California, Los Angeles (UCLA), explains "Most of the variability in images is irrelevant to the task. For instance, if you want to recognize a friend in a picture, you want to do so regardless of where she will be, how she will be dressed, whether she is partially occluded, what sensor will be used for the picture, etc. If you think of all the possible images of your friend, they are, for all practical purposes, infinite. So if you wanted a minimal representation—something that distills the essence of 'your friend' in every possible future image of her—that should be a much, much smaller object than an image."

Unfortunately, networks can home in on details that are very different to those used by humans. This leads to sometimes intriguing failures. Researchers at Kyushu University in Japan discovered late last year that modification of just one individual pixel in an image could upset neural networks trained to classify objects and animals; a taxi might suddenly be misidentified as a dog with such a tiny change.

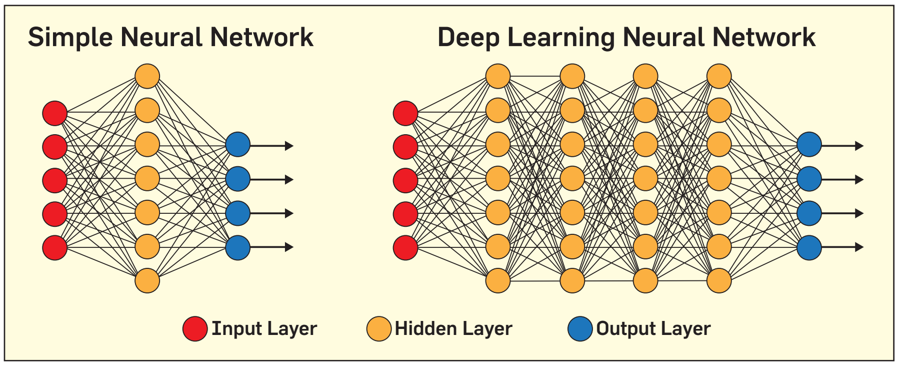
▸ "Trained neural networks can be tricked to focus on patterns in images that are barely noticeable by humans into a situation where they completely misinterpret the contents," says Chiyuan Zhang, a researcher working at the Center for Brains, Mind and Machines based at the Massachusetts Institute of Technology (MIT). "This leads to security concerns. It could potentially be used to implant backdoors in neural network models in ways that are hard to identify."

Mathematical interpretations of how deep neural networks learn offer one path to understanding why they generalize so effectively, and may provide mechanisms for them to avoid training on the wrong types of feature. Researchers regard the layering used by deep learning as one vital attribute. The layers make it possible to pull identifying marks out of images no matter where they are within the sample.

However, that is only part of the problem.

Tomaso Poggio, principal investigator at the McGovern Institute for Brain Research based at MIT, says, "It is important to understand there is much more work to be done [in deep learning]. Our hope is that if we understand better how they work we will understand better how they fail and, by doing that, improve them."

One strand of math-oriented research focuses on information theory. Naftali Tishby of the Hebrew University



A simple neural network has up to two layers hidden between the input and output layers; more than that, and it becomes a Deep Learning Neural Network, which can model complex non-linear relationships.

IMAGE COURTESY OF XENONSTACK.COM/BLOG

of Jerusalem believes the training processes in neural networks illustrate a branch of information theory that he helped develop two decades ago. He coined the term "information bottleneck" to describe the most efficient way that a system can find relationships between only the pieces of data that matter for a particular task and treat everything else within the sample as irrelevant noise.

Tishby's hunch was that neural networks provide examples of the information bottleneck at work. He worked with colleague Ravid Shwartz-Ziv to build a simpler form of neural network able to demonstrate how the process works. First the network finds important connections by adjusting the weights that neurons use to determine which of their peers in the network should have the greatest influence. Then, the network optimizes during what Tishby calls the compression phase. Through this process, neurons adjust weights to disregard irrelevant inputs. These inputs might represent the backgrounds of images of animals presented to a network trained to classify breeds using visual features.

However, an attempt last autumn by an independent team to replicate the results obtained by Tishby and Shwartz-Ziv using techniques employed by production neural networks failed to yield the expected compression phase consistently. Often, a neural network will achieve peak performance some time before it moves into the phase that Tishby refers to as compression, or may simply not follow the same pattern. Yet, these networks exhibit the generalization capability that the information bottleneck concept predicts. "I think the information bottleneck may be wrong or, in any case, unable to explain the puzzles of deep nets," Poggio says.

Poggio and colleagues look at the problem of understanding deep learning from the perspective of it being a process of iterative optimization. In learning what is important from the training data, the network arranges itself to minimize an error function—an operation common to optimization functions. In practice, the error functions for neural networks for a given set of training data seem to exhibit multiple "degenerate" minima,

The secret to deep learning's success in avoiding the traps of poor local minima may lie in a decision taken primarily to reduce computation time.

which seem to make it easier to find good solutions that generalize well. However, away from these wide valleys that lie toward the bottom of the error function's landscape, there are countless local minima that could trap an optimizer in a poor solution.

The secret to deep learning's success in avoiding the traps of poor local minima may lie in a decision taken primarily to reduce computation time. After each pass through the training set, the backpropagation algorithm that tunes the weights used by each neuron for the next test should analyze all of the data. Instead, stochastic gradient descent (SGD) uses a much smaller random sample that is far easier to compute. The simplification causes the process to follow a more random path towards the global minimum than full gradient descent. A result of this seems to be that SGD can often skip over poor local minima.

"We are looking for a minimum that is most tolerant to perturbation in parameters or inputs," says Poggio. "I don't know if SGD is the best we can do now, but I find almost magical that it finds these degenerate solutions that work."

For Soatto and his UCLA colleague Alessandro Achille, more clues as to how to make neural networks work better will come through studies that use the concept of the information bottleneck theory to look at the interactions between different network architectures and the training data.

Says Soatto, "We believe [Tishby's] ideas are substantially correct, but there are a few technical details that have to be worked out. The fact that we converged to similar ideas is remarkable because we started from completely independent premises."

Achille and Soatto used ideas from the information bottleneck to develop training optimizations that help smaller networks tune out noise. Timing also appears to be important, they believe. One 2017 experiment performed with Matteo Rovere of the Ann Romney Center for Neurologic Diseases in Boston, MA, indicated there is a critical phase early in training that proves crucial when the network weights are easily changed and the relationships between neurons quite plastic. The early phase has similarities to that proposed by Tishby and Shwartz-Ziv. Once this phase takes place, it seems to bias the network toward finding good minima as optimization proceeds.

Although the work on the information bottleneck and on optimization theory is beginning to lead to a better understanding of how deep learning works, Soatto says, "Most of the field is still in the 'let all the flowers bloom' phase, where people propose different architectures and folks adopt them, or not. It is a gruesome trial-and-error process, also known as 'graduate student descent', or GSD for short. Together with SGD, these are the two battle-horses of modern deep learning." ◼

Further Reading

Shwartz-Ziv, R., and Tishby, N.
Opening the Black Box of Deep Neural Networks via Information. ArXiv: https://arxiv.org/abs/1703.00810

Zhang, C., Bengio, S., Hardt, M., Recht, B., and Vinyalis, O.
Understanding Deep Learning Requires Rethinking Generalization. ArXiV: https://arxiv.org/abs/1611.03530

Poggio, T., Liao, Q., Miranda, B., Rosasco, L., Boix, X., Hidary, J., and Mhaskar, H.
Theory of Deep Learning III: Explaining the Non-Overfitting Puzzle. *CBMM Memo 073* (2017). https://cbmm.mit.edu/sites/default/files/publications/CBMM-Memo-073.pdf

Achille, A., Rovere, M., and Soatto, S.
Critical Learning Periods in Deep Neural Networks. UCLA-TR-170017. ArXiV: https://arxiv.org/abs/1711.08856

Chris Edwards is a Surrey, U.K.-based writer who reports on electronics, IT, and synthetic biology.

Technology | DOI:10.1145/3204449

Keith Kirkpatrick

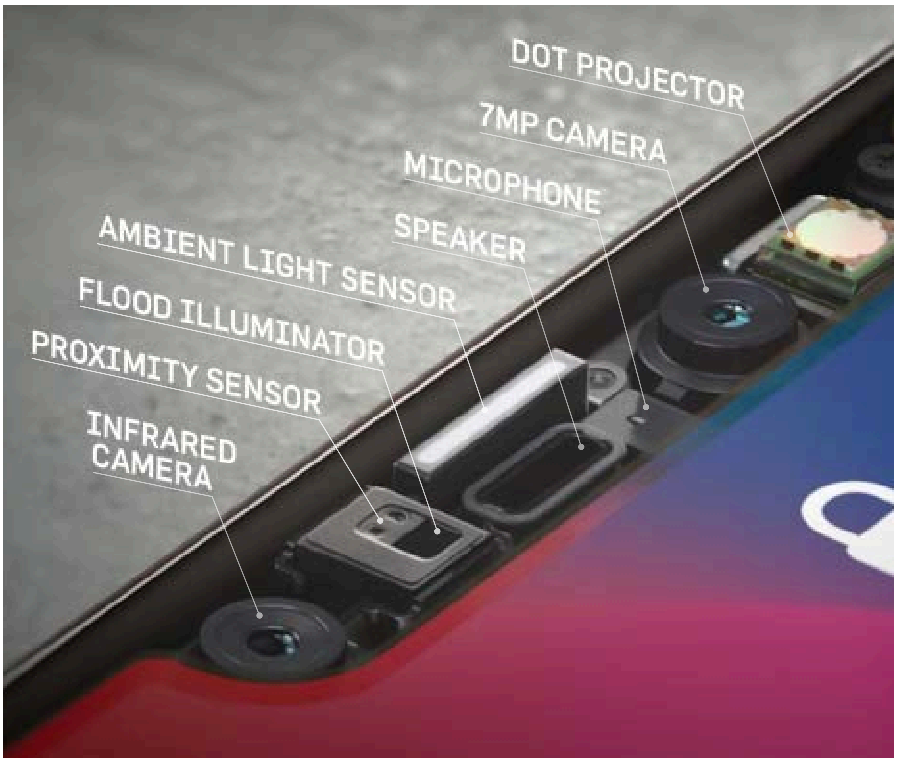
3D Sensors Provide Security, Better Games

A variety of techniques allow sensors to locate and recognize objects in space.

SENSOR TECHNOLOGY IS designed to allow machines to interact with real-world inputs, whether they are humans interacting with their smartphones, autonomous vehicles navigating on a busy street, or robots using sensors to aid in manufacturing. Not surprisingly, three-dimensional (3D) sensors, which allow a machine to understand the size, shape, and distance of an object or objects within its field of view, have attracted a lot of attention in recent months, thanks to their inclusion on Apple's most-advanced (to date) smartphone, the iPhone X, which uses a single camera to measure distance.

Indeed, the TrueDepth system, which replaces the fingerprint-based TouchID system on the Apple handset, shines approximately 30,000 dots outward onto the user's face. Then, an infrared (IR) camera captures the image of the dots, which provides depth information based on the density of the dots (closer objects display a dot pattern that is spread out, whereas objects that are farther away create a denser pattern of dots. Altogether, the placement of these dots creates a depth map with 3D data that is used to supply the system with the information it needs to check for a facial identity match, which then unlocks the device. The key advantage to this single-camera system and others like it is the relatively low cost of implementation.

For many consumers, their first exposure to 3D sensing technology came in 2010, in the form of Microsoft's groundbreaking Kinect motion-sensing input device designed for the Xbox gaming consoles and Windows PCs. Utilizing a chipset from Israeli developer PrimeSense (which was since purchased by Apple in November 2013), the 3D scanner system called Light Coding incorporated an IR emitter and an IR depth sensor. The emitter projected infrared light beams, and the depth sensor read those IR beams that were reflected back to the sensor. The reflected beams were converted into depth information that allowed the distance between an object and the sensor to be measured. The result was a gaming system that could accurately track a player's motion, so long as they stayed within an approximately six-square-meter zone in front of the Kinect bar.



The components in Apple's iPhone X required for Face ID 3D-scanning technology to work.

In subsequent versions of the Xbox, Microsoft replaced the PrimeSense technology with its own time-of-flight sensor technology, which features wide-angle coverage and three times the fidelity of the previously used technology. Not surprisingly, as the technology has improved, so has the demand for devices with embedded 3D sensing technology, according to Anand Joshi, a principal analyst with technology research firm Tractica LLC.

"Cameras have become really cheap and are being integrated into a wide range of devices," Joshi says, noting that the growth and availability of computer vision technology and the software to extract depth information has enabled a large application development ecosystem to thrive, thereby supporting even more innovations.

Another 3D imaging technique that may support more advanced functionality is called stereopsis, or stereovision. This technology uses a sensor that takes the feeds from two cameras, and then compares the delta between the horizontal placement of each object to determine how far the object is from the sensor. This technique is deployed by LinX, another technology company acquired by Apple, possibly because the technology is not subject to interference, portending greater potential for use in outdoor applications and over greater distances.

While unconfirmed, it is possible this stereo vision technology may be used within the iPhone's forthcoming rear 3D sensor package, which may add 3D depth-sensing features to the handset's rear camera, possibly enabling virtual and augmented reality applications, along with new types of videoconferencing and gaming applications, to be accessed via the iPhone.

Other 3D sensors utilize a variety of techniques to recognize objects in space, although they each use beams of light to measure distance and therefore may also be used to measure or ascertain the size, shape, and orientation of an object. Called "time of flight" (TOF) technology, it is based on a laser scattering infrared light pulses; the technology measures the amount of time it takes for the pulses to bounce back, to ascertain where an object is located. A key advantage of this technology is that an object's proximity and size can be captured very quickly with a high degree of accuracy (within a few centimeters), making it suitable for industrial, automotive, or other applications where speed and accuracy are required.

Examples of sensors that use time-of-flight technology include light detection and ranging (LiDAR) sensors used in advanced driver assistance systems (ADAS) to determine the distance between the vehicle and other objects (for systems such as forward collision warning, lane-departure warning, and adaptive cruise control), as well as in gaming systems like the second-generation Kinect sensor, which is used in Microsoft's popular Xbox One console. LiDAR technology is being incorporated across the board by automotive OEMs, as they seek to incorporate sensing technology that can augment and enhance drivers' awareness and reaction times. The advanced sensing technology is also being incorporated into forthcoming autonomous vehicles as part of a comprehensive and integrated network of sensing technology.

Intel Corp's RealSense technology functions similarly to the Kinect's cameras, by projecting a laser and measuring how it bounces or otherwise interacts with the environment. This type of technology is well suited to indoor applications, and is well supported by an existing base of software designed specifically to manage this technique..

The choice of the type of sensor to utilize is largely based around the application, and how the technology will be deployed, according to Joshi.

"One can use single, stereo, or a multiple-camera based system," he explains. "You can also supplement the visible light image with additional sensors such as infrared or radar. Mobile phones are based on a single or stereo camera, and use TOF or triangulation. However, automotive [applications] use radar or other sensors to supplement camera systems. [Microsoft's game-focused] Kinect has used structured light. The choice of algorithm is up to the developer, depending on the sensors and compute capacity available."

Joshi notes that while single-camera 3D systems, such as those used in mobile phones, are least expensive because they are limited. Says Joshi, "The sophistication starts when you add stereo cameras, which use two sensors, and keeps going on up when IR and other sensors are added."

Applications for 3D sensors are not limited to smartphones; they can be used for any type of 3D scanning in which an object must be mapped with precision.

The growth of the personal mobile handset market, as well as other verticals, including automotive, healthcare, and retail, has led to significant activity among makers of 3D sensors and chipsets. Indeed, Sony Corp., which currently has 49% of the market for image sensors, is developing new TOF sensors that are smaller than today's sensors, and can calculate depth at greater distances.

Meanwhile, STMicroelectronics, which currently supplies Apple and a number of other smartphone vendors with its FlightSense sensors, has shipped more than 300 million TOF chips, according to the company. These sensors are suited for a number of applications beyond smartphones, including drones, augmented-reality applications, and industrial applications. A single-module design integrating a laser and sensor array allows reliable proximity detection, ambient light detection, and low-power operation to be easily incorporated into a number of device types.

Not to be outdone, Apple announced it has invested $390 million out of its $1-billion Advanced Manufacturing Fund in Sunnyvale, CA-based Finisar Corp., a supplier of vertical-cavity surface-emitting lasers (VCSELs), helping to fuel speculation Finisar will ultimately wind up being a supplier involved with the potential rear-facing 3D sensor system on the next iPhone. Beyond basic scanning, the real value of 3D sensors is the ability to combine them with other sensor types. Industry participants and analysts say the availability of lower-cost, higher-powered processors and software has enabled this sensor data fusion, which combines input from multiple sensors and then processes the data in real time, that is enabling devices to be built with near-human-like sensing abilities.

"Computer vision has also allowed sensor fusion to incorporate additional sensors such as laser and radar," Joshi says, noting that the market has been driven by a "a combination of falling price [and] better technology" availability.

According to Joshi, Intel, Google, Apple, and Microsoft are leading the way on the consumer side, while Natick, MA-based Cognex Corp. is big on the industrial side. Within the automotive sector, Nvidia is the leader in AI chipsets that are used to power computer vision technology, and many automotive companies use them along with a wide variety of other sensors to enable autonomous driving and advanced driver-assistance systems.

However, applications for full-sized 3D sensors are not limited to smartphones or other consumer devices; the technology can be used for any type of 3D scanning in which an object must be mapped with precision. Applications exist or are in the works involving 3D printing, design, mapping, object recognition, facial recognition, gesture-based control, and other industrial or commercial applications.

Jae-Yong Lee, a venture capital investment manager with ReWired, a $100-million robotic-focused venture studio based in London, says he sees great potential for the technology in retail analytics. "If you could install affordable [3D] sensors within a store so you could really measure the traffic—where the people are, where they're walking toward, where they are hesitating—you could really optimize how you display your items and how you can redesign the store for more sales."

In the near term, the potential for 3D sensors beyond consumer devices is likely to develop within the burgeoning autonomous vehicle market, says Pier-Olivier Hamel, a product manager with Leddartech, a Quebec, Canada-based company that manufactures LiDAR technology for the automotive market. The company's technology is currently focused on using LiDAR to accurately capture the 3D profile of objects perceived by the sensor, and making this technology available to automobile OEMs and Tier-One suppliers to classify objects that can be used to train autonomous and semi-autonomous driving systems. While the company is not focusing on creating LiDAR technology for smaller devices now, it certainly is a possibility for the future.

"We can see with the miniaturization of everything," Hamel explains. "I'm sure that there will be applications for personal devices and smartphones or augmented reality; those are all possibilities."

In the near term, the potential for 3D sensors beyond consumer devices is likely to develop in the burgeoning autonomous vehicle market.

With all this activity, it is no surprise the market for 3D sensors and components is likely to rise. Tractica, which released its 3D Imaging Hardware and Software market study in early 2016, projected the market for 3D imaging technology would reach $24.9 billion globally by 2024, up from $3.2 billion in 2014, which reflects a compound annual growth rate (CAGR) of 23%.

Joshi, the study's author, says that while the $24.9-billion market value may be overly bullish, as the technology did not take off quite as quickly as originally predicted, the consumer and mobile market will continue to be the largest sectors for 3D imaging technology, accounting for about $10.1 billion of the overall market by 2024. C

Further Reading

3D Imaging Hardware and Software Market to Reach $24.9 Billion by 2024, Tractica LLC, January 14, 2016
http://bit.ly/2BDJIrF

Graham, L.A., Chen, H., Cruel, J., Guenter, J., Hawkins, B., Hawthorne, B., Kelly, D.Q., Melgar, A., Martinez, M., Shaw, E., and Tatum, J.A.
High-power VCSEL arrays for consumer electronics
http://bit.ly/2BECC69

What Is Time-of-Flight? – Vision Campus, Basler AG, May 31, 2016
http://bit.ly/2BDKLrB

Keith Kirkpatrick is principal of 4K Research & Consulting, LLC, based in Lynbrook, NY, USA.

ACM Member News

A FASCINATION FOCUSED ON SUSTAINABILITY

"I have always been inclined toward math, and was fascinated by computers very early on," says Carla Gomes, a professor of computer science and the director of the Institute for Computational Sustainability at Cornell University in Ithaca, NY. "I have always known I would be doing something around this."

Gomes earned her master's degree in applied mathematics from the Technical University of Lisbon, Portugal in 1987, and her Ph.D. in computer science from the University of Edinburgh, U.K., in 1993. She did postdoctoral research at the U.S. Air Force Research Laboratory for five years before joining Cornell University in 1998, where she has been ever since.

Her main area of interest lies at the intersection of artificial intelligence (AI) and operations research. Currently, her focus is on computational sustainability, an emerging field that studies and develops solutions to computational problems for balancing environmental, economic, and societal needs for a sustainable future. Gomes feels AI and computer science can make a real difference in addressing sustainability challenges.

She explains, people often think of sustainability as referring solely to the environment, but sustainable development is about balancing environmental, economic, and societal issues, with the ultimate goal of assuring the well-being of current and future generations. Gomes is passionate about developing intelligent systems that can help across a variety of domains to address this.

"If you inject computational thinking and methodologies," Gomes says, "we can come up with creative solutions to some of the biggest challenges we are facing today."

—*John Delaney*

Society | DOI:10.1145/3204447

Logan Kugler

Getting Hooked on Tech

Are technology companies maximizing profits by making users addicted to their products?

It is not a good year for Facebook. The company's ubiquitous platform, designed to bring users together, was used by Russian non-state actors to tear America apart by creating fake posts on highly divisive issues and using them to sway opinion in the lead up to the 2016 presidential election.

In hindsight, Facebook was the perfect weapon: used by billions worldwide, and more than half of Americans use it several times each day. That much attention, it turns out, could be weaponized by the Russians.

According to *The New York Times*, "Facebook found $100,000 of ad purchases that were linked to the fake pages—designed to look like the pages of Americans animated by particular issues—that spread inflammatory messages about immigration, guns and other topics; derided Mrs. Clinton and supported Mr. Trump."

The impact of these efforts was perceived as so dire that Facebook agreed to turn over to Congress more than 3,000 ads used to influence attitudes during the election. At the time, Facebook founder and CEO Mark Zuckerberg said, "It's a new challenge for Internet communities to deal with nation-states attempting to subvert elections. But if that's what we must do, we are committed to rising to the occasion."

That's nice, but only tells one side of the story.

While the fake posts were designed to provoke outrage, some users kept coming back multiple times per day; they were shown even more fake news, and further influenced by false narratives.

If these types of sensational posts are like a drug, then users have been programmed to be unable to get enough of them from social media sites and the smartphones that display them.

How Tech Gets You Addicted

Facebook is just one of several companies that rely on users' continued attention to make money. Twitter and Google operate on this model, too. All three rely on advertising revenue to make money. The value of that advertising revenue is directly tied to the attention the platform can attract and keep from users.

Tech giants excel at getting attention, offering products with features users can't access anywhere else. It might be the ability to search for any information in the world using Google's search engine, or the ability to communicate and engage with friends and family from anywhere in the world by using Facebook. Users might even want to engage in a real-time conversation with thousands of other people about breaking news by using Twitter.

However, it is not just the content that keeps users coming back for more, but also how the platforms are engineered to exploit human psychology. While the subjective condition of "using tech too much" is not classified as a biological addiction, it can be a behavioral one.

"Tech companies develop their products in order to make them appealing and user-friendly, so you're keen to use them," says Daria Kuss, a psychologist and senior lecturer at Nottingham Trent University. Kuss researches the psychology of Internet use, trying to decode how products and services are designed to keep users returning for more.

"Some ways in which tech companies engineer their devices is to allow personalization, in terms of content, appearance, and usage—tailoring the products to the respective needs of the users," she says.

"It's a social validation feedback loop ... exactly the kind of thing a hacker like myself would come up with."

Facebook and Google lead the way in this arena. Facebook allows users to customize their personal profile to their liking and indicate their interests by engaging with content. Every reaction a user has to a post teaches Facebook's algorithms his/her preferences; these algorithms then serve more content that even better matches the user's preferences.

The result? The user visits and revisits the site, staying for longer, making them a richer target for advertising.

Google does something similar, trying to guess which search results will best match a user's query, then serving them up with increasing accuracy based on insights that machine learning algorithms extract from its immense pool of the world's search data. Companies pay Google to display their ads next to those increasingly relevant results.

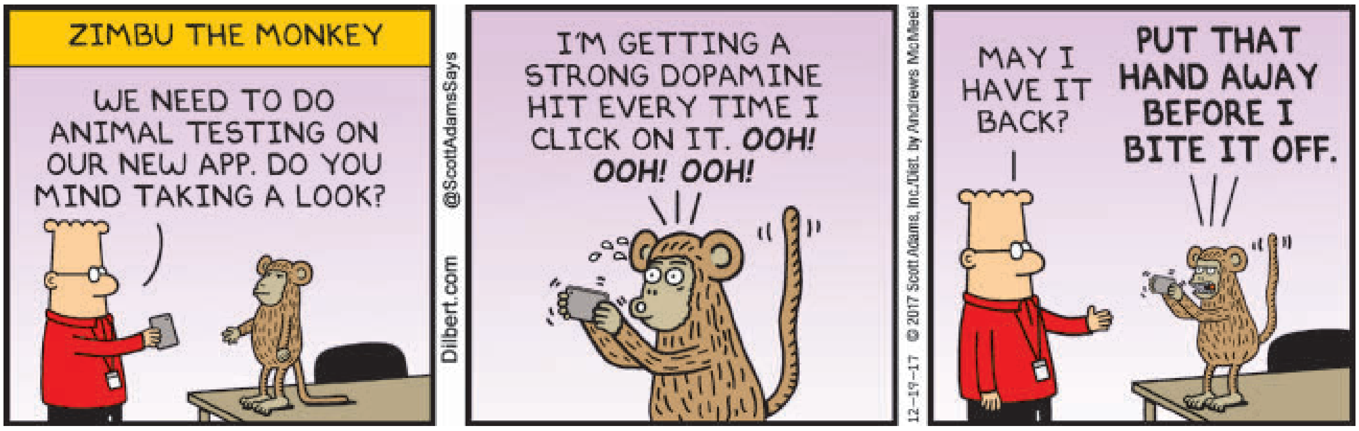
There certainly is nothing wrong with providing more of what users want, but tech giants include their fair share of socially and psychologically engineered tweaks to keep users coming back for more, and it goes beyond making the service more useful.

Sean Parker, a former president of Facebook, has recently become what he calls a "conscientious objector" on social media, speaking out against the negative effects of the platform he helped to create. He told *Axios* that the process behind Facebook and other social media websites is wholly concerned with consuming as much of users' time and attention as possible.

This means these companies "need to sort of give you a little dopamine hit every once in a while, because someone liked or commented on a photo or a post or whatever," Parker said. "And that's going to get you to contribute more content, and that's going to get you more likes and comments.

"It's a social validation feedback loop ... exactly the kind of thing that a hacker like myself would come up with, because you're exploiting a vulnerability in human psychology."

This also happens on smartphones,


ZIMBU THE MONKEY
WE NEED TO DO ANIMAL TESTING ON OUR NEW APP. DO YOU MIND TAKING A LOOK?
I'M GETTING A STRONG DOPAMINE HIT EVERY TIME I CLICK ON IT. OOH! OOH! OOH!
MAY I HAVE IT BACK?
PUT THAT HAND AWAY BEFORE I BITE IT OFF.
Dilbert.com @ScottAdamsSays
12-19-17 © 2017 Scott Adams, Inc./Dist. by Andrews McMeel

where notifications train user to look because they have received a text message, social media update, or news alert.

Kuss details how this type of functionality manifests in games that you are unable to put down. "Many games use rewards that are delivered based on intermittent reinforcement schedules, i.e., rewards are provided only some of the times the gamer is performing the action, making it more likely for them to continue engaging in the behavior."

Except this is no game; it's a battle for your waking life—and one that can have negative effects on your health.

Hazardous to Your Health?

Caglar Yildrim, a professor of human-computer interaction at the State University of New York at Oswego, believes smartphone addiction is a real problem. In his research, Yildrim uses a questionnaire that scores people on the severity of their smartphone dependence. If you score between 100 and 200, you may actually experience severe anxiety when parted from your device.

"This might negatively affect your social life and relationships with friends and family," says Yildrim. "There are studies that show those who score high on the test tend to avoid face-to-face interactions, have high levels of social anxiety, and maybe even depression. It might affect your ability to work or study, because you want to be connected to your smartphone all the time."

One of the biggest dangers presented by addictive tech is distracted driving. According to the *Pew Research Center*: "In a 2010 Pew Research Center survey, nearly half (47%) of adults who use text messaging (equivalent to 27% of all U.S. adults) said they had sent or received messages while driving."

CNN reports nearly half of U.S. adults say they have sent a text while driving; distracted driving kills nine people and injures more than 1,000 every single day.

When addictive tech starts to override rational precautions like paying attention while driving, the consequences could be injury or death.

The mental toll of addictive services like Facebook can't be discounted, either. Another former Facebook employee, former VP of user growth Chamath Palihapitiya, said he does not allow his children use that social media service because he fears they will become addicted to "short-term, dopamine-driven feedback loops."

Kuss and a colleague analyzed previous studies on social media addiction in 2011; they concluded social media addiction actually "may be a potential mental health problem for some users."

To date, social media addiction is not formally classified as a mental health disorder. Just how much damage has already been done is impossible to tell, but the fact remains that many users check these services, almost unconsciously, many times a day—and this can have extremely negative consequences to time and health.

So what can users do to reduce their dependence on services and devices intentionally designed to be addictive?

"Personally, I would recommend, counterintuitively, to make time for technology use," says Kuss. "This takes the immediate pressure off to use technology." She recommends an hour in the morning and an hour at night, then setting the smartphone aside.

"I would also recommend putting the technology away when having dinners with the family and spending time in face-to-face interactions with others."

Perhaps the most important step, however, is truly understanding there is a problem, she says.

"Overall, we need to create an increased awareness of our technology use. Our phones allow us to check the time we spend on specific applications and I recommend having a look at this—it's quite enlightening. The time we spend using these apps is often longer than we think." C

Further Reading

Cakebread, C.
If you haven't checked Facebook lately, you're in the minority,
***Business Insider*, Oct. 2, 2017,**
http://read.bi/2AT5WC4

Shane, S., and Isaac, M.
Facebook to Turn Over Russian-Linked Ads to Congress,
***The New York Times*, Sept. 21, 2017,**
http://nyti.ms/2yS3wCb

Sean Parker unloads on Facebook "exploiting" human psychology,
***Axios*, Nov. 9, 2017**
http://bit.ly/2kg5Pdx

LaMotte, S.
Smartphone addiction could be changing your brain,
CNN, Dec. 1, 2017
http://cnn.it/2kHY13J

Texting while driving may be common, but it's illegal in most states,
***Pew Research Center*, Nov. 15, 2013**
http://pewrsr.ch/2kFnNFZ

Online Social Networking and Addiction—A Review of the Psychological Literature, International Gaming Research Unit, Nottingham Trent University, Aug. 29, 2011
http://bit.ly/2J6TKBo

Logan Kugler is a technology writer based in Tampa, FL, USA. He has written for over 60 major publications.

DOI:10.1145/3208095

Nicholas Weaver

▸ Peter G. Neumann, Column Editor

Inside Risks

Risks of Cryptocurrencies

Considering the inherent risks of cryptocurrency ecosystems.

CRYPTOCURRENCIES, ALTHOUGH A seemingly interesting idea, are simply not fit for purpose. They do not work as currencies, they are grossly inefficient, and they are not meaningfully distributed in terms of trust. Risks involving cryptocurrencies occur in four major areas: technical risks to participants, economic risks to participants, systemic risks to the cryptocurrency ecosystem, and societal risks. Fortunately, for all but the last case, there is little risk to anyone not directly participating; (see the article "Privacy in Decentralized Cryptocurrencies" on page 78 in this issue).

Cryptocurrencies are tradeable cryptographic tokens, with Bitcoin as the most famous example. Bitcoin, developed by a pseudonymous creator, Satoshi Nakamoto, consists of a distributed public ledger system showing all balances associated with public keys. To spend Bitcoin, someone with the corresponding private key signs a message indicating the particular balances should be transferred to a set of destinations and then broadcasts this message through a peer-to-peer network.

The primary notion behind Bitcoin's design is to enable a censorship-resistant and irreversible payment system.

This peer-to-peer (P2P) network then validates the transaction as valid in the public ledger and commits it to another block in the ledger containing at most 1MB of data. In order to prevent the block from being tampered, the Bitcoin system uses "proof of work"[1] to protect its hash chain. Each block contains a pointer to the previous block (creating the "blockchain"), and every miner attempts to create a new valid block by computing partial hash collisions, hoping to find a value less than a dynamically tuned difficulty factor by taking a potential block and modifying various mutable fields until the miner discovers a partial collision that meets the difficulty requirement.

Once a miner discovers a new block, it broadcasts this block over the peer-to-peer network; all other miners then validate the new block and start mining the next block. As a consequence, changing the last *n* blocks in the ledger requires approximately the same number of hash calculations as creating those *n* blocks. Each block also contains a transaction that pays a fixed reward to the winning miner, as well as all transaction fees sent in the block. In order to implement a fixed monetary policy, the difficulty factor self-adjusts on regular intervals to limit the block creation rate to one block approximately every 10 minutes, and the block reward halves approximately every four years.

This naturally creates a "Red Queen's Race," which currently causes the Bitcoin network to consume more power than Ireland. When there is

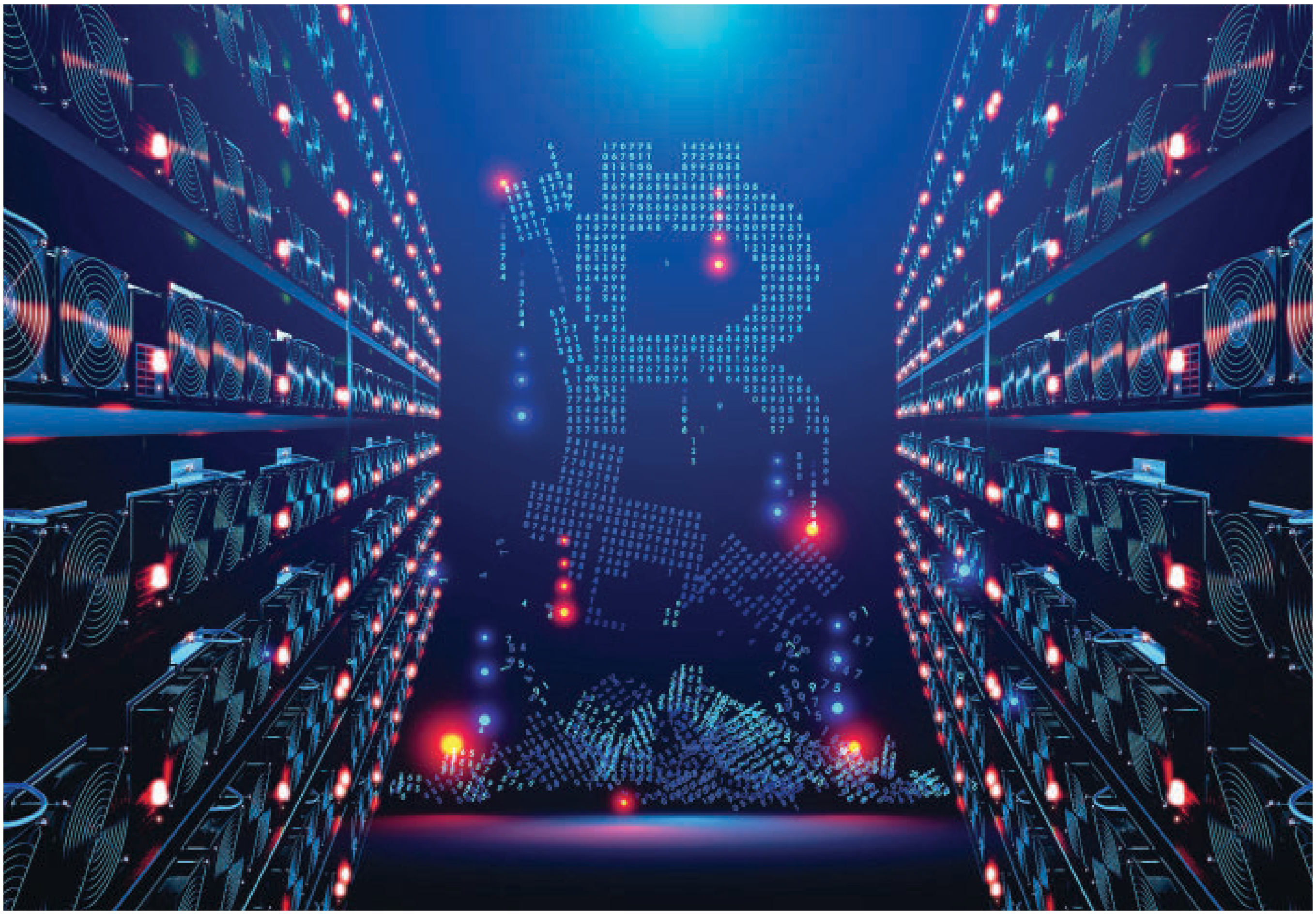


IMAGE COLLAGE BY ANDRIJ BORYS ASSOCIATES/SHUTTERSTOCK

potential profit, more miners are incentivized to join the process until the point where nobody makes a profit anymore. For example, a 10x reduction in power consumption per hash for Bitcoin mining would have little real effect on Bitcoin's power consumption. Instead, there would just be 10x as many hash computations needed to produce a block.

A good rule of thumb is that when prices are stable, approximately one-third to one-half of the block reward is sold by miners to pay power bills. This implies that when prices are high, Bitcoin consumes an outrageous amount of power. Any system based on proof-of-work will suffer this fate: If there is profit in mining, the miners will keep using more and more power until there is no more excess profit available. The only way Bitcoin could reduce its power consumption is through a massive collapse in price.

The fixed block size also limits transaction throughput to a trivially small global rate that is approximately three transactions per second. Although transaction fees start low, they can quickly increase when the transaction rate exceeds the global limit—as only those willing to pay increasing auction-based fees see their transactions confirmed. This is what caused the recent spike in Bitcoin transaction costs to a median price of over $30 a transaction. These global volume limits make Bitcoin clearly unsuitable as a public ledger. Nevertheless, a comparable cryptocurrency that supported 300 inexpensive transactions per second could see its global state grow at an untenable 14GB/day of additional storage for every participating node in the network, storage that also needs to be searched to validate transactions.

Since the original deployment of Bitcoin, a host of other cryptocurrencies has arisen, often by simply modifying the Bitcoin source code and changing a few parameters. These have taken many forms, including faster-committing blocks with a catchy slogan (Litecoin: "Litecoin is silver to Bitcoin's gold."), explicit jokes (Dogecoin), forks that maintain the same history until the date of the fork (Bitcoin Cash), and some notable ideas including an attempt to create a stable reserve of value by reinventing 18th-century banking (Tether) and "programmable money" to create smart contracts (Ethereum).

Cryptocurrencies for Payments: Not Fit For Purpose

The primary notion behind Bitcoin's design is to enable a censorship-resistant and irreversible payment system. It is intended that there should be no central authority that can say "thou shalt not" or "thou shouldn't have." The only other analogue in the real world is cash, which is bulky and requires physical presence.

All other electronic payment systems have the potential for censorship. There are third parties involved in the payment process that, under government pressure, can and do seek to ban or reverse disallowed payments. This includes blocking a wide assortment of criminal activity, such as drug payments, ransom and extortion payments, and money laundering. It can also be used to implement currency controls (limiting the ability of residents to exchange local currency for dollars or euros) and conduct financial blockades, such as how U.S. credit cards cannot be used for online gambling, could not be used to buy ads on Backpage (when this now-admitted criminal enterprise was first blocked from accepting credit cards due to local pressure), or transfer money directly to WikiLeaks.

However, unless censorship resistance in an electronic transaction is a requirement (such as for drug deals, ransom payments, money launderers, and those seeking to evade currency control), irreversibility combined with the volatile price means Bitcoin is significantly inferior to alternatives such as credit cards or PayPal.

Most sensible recipients of a Bitcoin payment immediately convert their payment into dollars, to avoid the substantial risk that currency swings may prove costly. Thus, most legal sites that accept Bitcoin payments are not actually taking Bitcoin, but instead using a service that both adjusts the Bitcoin price dynamically (so the merchant is actually pricing in U.S. dollars) and immediately sells the Bitcoin.

This also means that unless the buyer is a believer in Bitcoin, the buyer ought to buy Bitcoin only immediately before they initiate the transaction, to avoid volatility (and will have had to mine or buy the Bitcoin in any case). This is the point where Bitcoin's irreversibility results in substantial costs.

The Bitcoin exchange either effectively has to take cash only, must wait several days after a bank transfer completes before allowing the customer to buy Bitcoin, or is implicitly extending credit to the customer. Any exchange that does not follow these rules faces the fate of Tradehill, a Bitcoin exchange that went defunct when faced with chargebacks on Dwolla-based bank transfers. Steve Wozniak recently experienced the same fate when he sold $75,000 in Bitcoin to an individual who paid with a credit card, only to find the transaction canceled since the thief used a stolen credit card (see https://cnb.cx/2EUxVY6).

Most sensible recipients of a Bitcoin payment immediately convert their payment into dollars.

Bitcoin payments are thus significantly more expensive for legal purposes when including the mandatory two currency conversion steps, the first one of which must be either slow, involve cash, or an implicit extension of credit. Even eliminating the irreversibility (which goes contrary to a fundamental explicit Bitcoin design goal stated by Nakamoto) would still result in two currency conversion steps. It is impossible to eliminate these two steps from a volatile cryptocurrency.

Yet, for those who do believe in Bitcoin it still is not usable as a currency. The monetary policy for Bitcoin is fixed with a limited and prescheduled creation rate designed to be deflationary. The only rational behavior for someone holding a deflationary currency is to never actually spend it. Otherwise the person risks eternal regret for buying a 10,000 BTC pizza (in 2010) and later realizing the pizza's payment is now worth a notional $100M.

Individual Technical Risks

Since cryptocurrencies are controlled by private keys, anyone who gains access to the private key can move the currency. This makes cryptocurrencies incredibly vulnerable to theft. If someone holds their cryptocurrency using a third-party service, they run the continual risk that the service gets robbed—an almost routine occurrence throughout the short history of Bitcoin. Thus, users instead need to store their money on their own systems.

But even this is difficult. During early research into Bitcoin when attackers installed Bitcoin miners on compromised systems, we hypothesized that malcode might also start to include Bitcoin theft amongst the automatic functionality. So we created a small Bitcoin wallet, placed it on images in our honeyfarm, and set up monitoring routines to check for theft. Two months later our monitor program triggered when someone stole our coins.

This was not because our Bitcoin was stolen from a honeypot, rather the graduate student who created the wallet maintained a copy and his account was compromised. If security experts can't safely keep cryptocurrencies on an Internet-connected computer, nobody can. If Bitcoin is the "Internet of money," what does it say that it cannot be safely stored on an Internet connected computer?

Bugs can also naturally cause significant damage to cryptocurrency holdings. Although this potentially can affect any cryptocurrency, the biggest danger for bugs arises when cryptocurrencies are combined with "smart contracts"—programs that are generally immutable once deployed and that automatically execute upon the transfer of currency. The most successful platform for these is Ethereum, a cryptocurrency that allows writing programs in a language called Solidity.

Bugs in these smart contracts can be catastrophic. The first big smart contract, the DAO or Decentralized Autonomous Organization, sought to create a democratic mutual fund where investors could invest their Ethereum and then vote on possible investments. Approximately 10% of all Ethereum ended up in the DAO before someone discovered a reentrancy bug that enabled the attacker to effectively steal all the Ethereum. The only reason this bug and theft did not result in global losses is that Ethereum developers released a new version of the system that effectively undid the theft by altering the supposedly immutable blockchain.

Since then there have been other catastrophic bugs in these smart contracts, the biggest one in the Parity Ethereum wallet software (see https://bit.ly/2Fm7je4). The first bug enabled the mass theft from "multisignature" wallets, which supposedly required multiple independent cryptographic signatures on transfers as a way to prevent theft. Fortunately, that bug caused limited damage because a good thief stole most of the money and then returned it to the victims. Yet, the good news was limited as a subsequent bug rendered all of the new multisignature wallets permanently inaccessible, effectively destroying some $150M in notional value. This buggy code was largely written by Gavin Wood, the creator of the Solidity programming language and one of the founders of Ethereum. Again, we have a situation where even an expert's efforts fell short.

The entire cryptocurrency environment also faces systemic risks.

Individual Economic Risks

Everything about the cryptocurrency space is full of bubbles. Since all volatile cryptocurrencies are actually substantially inferior for legal purposes, this implies that the actual value as currency is effectively $0, so the only store of value is in other utility for a distributed trustless public append-only ledger.

Yet the Bitcoin blockchain, due to consolidation of mining into a few mining pools, does not actually distribute trust. Instead the system is effectively controlled by less than 10 entities self-selected by their willingness to consume power and anyone using Bitcoin implicitly trusts a majority of these few entities. Every proof of work blockchain seems to experience similar consolidation as the more efficient miners inevitably drive out less efficient ones. Given the almost trivial cost of building a three-transactions-per-second distributed system with identified and trusted entities using cryptographic signatures instead of proof of work this suggests the utility value for these cryptocurrencies is also effectively $0. This means everyone participating in the cryptocurrency market is basing the value only on the price that somebody else will pay—no different from tulip bulbs or beanie babies—and are all vulnerable to substantial and sudden collapse.

But further magnifying the problem is a large number of scams. There is a current trend in "Initial Coin Offerings," mostly consisting of cryptographic tokens implemented on top of an existing cryptocurrencies such as Bitcoin or Ethereum. Although claiming to be crowd-sold tokens for purchase of future services, the tradeable nature of these tokens has resulted in their acting as unregistered securities in a bubble market. There are also organized groups conducting pump-and-dump schemes, complete with fancy websites, animated advertisements, and even placing paper advertisements in BART commuter trains in San Francisco, CA. This market developed largely in absence of regulation, although regulators like the U.S. Securities and Exchange Commission are finally starting to pay attention.

Likewise, not only is a bubble often a natural Ponzi scheme, there are many explicit or likely Ponzi schemes. In the early days of Bitcoin approximately 10% of all Bitcoin were invested in Bitcoin Savings and Trust, a Ponzi scheme run by a pseudonymous individual known to the community only as PirateAt40. The editor of *Bitcoin Magazine* at the time so much believed it was not a Ponzi scheme that he made side bets that it was not, using Bitcoin that he did not have, just before the scheme collapsed.

Even explicitly advertised Ponzi schemes see significant activity, such as the "Proof of Weak Hands', a Ponzi scheme implemented as an Ethereum smart contract. More than $1 million in notional value flowed into the scheme in the space of a few hours before the flow stabilized. Two days later, one bug froze the scheme (making withdraws impossible) before a second bug enabled a thief to take all the value.

Systemic Risks

The entire cryptocurrency environment also faces systemic risks including worms, exchanges, central authorities, and government intervention.

Peer-to-peer systems, and especially those written in unsafe languages such as C and C++, are particularly vulnerable to worms. A worm that can exploit a P2P node and then spread to all connected nodes takes approximately the same time to spread worldwide as a broadcast message in the same network. For cryptocurrencies that minimize the time required to send transactions, this would enable a worm to spread globally in a matter of seconds.

The ease of theft and the common practice of speculators using multiple cryptocurrencies create an incentive for thieves to deploy worms, because a worm could spread through one cryptocurrency's network and then steal all other cryptocurrencies accessible on the victim computers. For example, Dogecoin (coded in C++) has effectively received no updates in two years, yet this explicit joke still has a notional value (at time of writing) of over $550M and is the 27th-largest cryptocurrency. The odds of a wormable vulnerability in the P2P software are significant, especially when combined with the observation that Dogecoin is a fork of Luckycoin's source, which was itself a fork of Litecoin, itself a fork of Bitcoin. Security patches in any of the upstream cryptocurrencies can act as a guide for discovering exploits.

The exchanges themselves also create systemic risks. Almost all exchanges seek to avoid regulation, which means they implode with almost seeming regularity—usually due to a combination of theft and fraud. These exchanges may even participate in active market manipulation.

A previous Bitcoin bubble appears to have resulted from deliberate price manipulation on the MtGox Bitcoin exchange; the current bubble may be due to the Bitfinex exchange creating Tethers and then using them to buy cryptocurrencies. There are also credible allegations of exchanges enabling wash trading, spoofing, insider trading, and other market manipulations.

Finally, cryptocurrencies are actually vulnerable to intervention by central authorities. Although cryptocurrency advocates claim there is no central authority that can censor transactions, the common collectivization of mining into a few entities, combined with official distributions, means small groups can arbitrarily change the rules, and have done so in cases such as a bug-related hardfork in Bitcoin and the Ethereum rollback of the supposedly immutable DAO contract in response to the DAO theft. Both showed that central authorities exist for even the biggest cryptocurrencies and that these authorities can act arbitrarily to rewrite the rules. Such interventions have generally been benign; however, that such interventions are even possible negates the basic thesis that these currencies lack central authorities.

Governments can also intervene to effectively kill cryptocurrencies, should that be desired. The most effective mechanism is simply regulation. Cryptocurrencies have value only when they can be converted back to local currency. By effectively strangling the exchange process, governments can make cryptocurrencies unworkable. Already most exchanges are now cut off from banking, limiting the conversion opportunities. Similar face-to-face individual exchanges (such as those felicitated on LocalBitcoins) are inevitably running afoul of local money-service laws. Enforcing these laws could further limit convertibility.

If cryptocurrencies succeed, we can expect a great increase in criminal bandwidth.

Governments (or others with a substantial budget) can also attempt technical disruptions. The limited transaction capability can be exploited by a government purchasing a quantity of Bitcoin, and then creating useless transactions. The goal of such a spam campaign would not be simply to clog the network, but also to generate responding spam filters. As the spam campaign continues, the goal becomes to tune the spam so that the filters cause false positives. How can a cryptocurrency work if a non-trivial fraction of legitimate transactions are blocked by spam filters?

Risks to Society

The aforementioned risks are all limited to market participants, and result in various failures. But the greatest risk to society may come not from failures, but from success. Beyond the obvious externalities imposed by cryptocurrency mining (a stable doubling in Bitcoin's price will further double its power consumption), it is primarily criminals who regularly benefit from censorship-resistant payments.

In many cases the bandwidth limit for crime is not the crime itself, but the money laundering. For criminals, cash is censorship-resistant but requires proximity and mass with $1M U.S. weighing approximately 10kg. Euros are more compact, requiring only 1.7kg in 500€ notes for the same value, leading the European Central Bank to begin phasing out the 500€ note. Additionally, it is deliberately difficult to move significant quantities of cash into the rest of the banking system, as deposits over $10,000 or other features generate suspicious activity reports.

If cryptocurrencies succeed, we can expect a great increase in criminal bandwidth. The only reason why the online drug markets remained small (approximately $1M a day in sales despite existing for half a decade) is that Bitcoin and the other cryptocurrencies are like the classic corrupt poker game; yes, it's rigged, but it's the only game in town. A cryptocurrency that actually offered both real anonymity and acted as a store of value (eliminating the need to constantly shift between dollars) would see an explosion in this market.

But such uses would not be limited to criminal-to-criminal transactions but would also act as a vehicle for extortion. The first ransomware epidemic a few years ago offered a choice to victims, either Green Dot or Bitcoin, with almost every victim using the much easier Green Dot, where the victim could purchase a MoneyPak from a convenience store and provide the numbers to the extortionist. It was the U.S. Treasury pressure on Green Dot (to break up a money-laundering flow) that disrupted that epidemic. How much greater would the current ransomware epidemic be if it was easy for victims to pay? How much other criminal extortion would target ordinary citizens?

Conclusion

The risks in the cryptocurrency world are multifaceted and diverse, but fortunately most are limited to those who participate. This leads to a natural conclusion. As the philosopher WOPR said in the movie *WarGames*, "The only winning move is not to play." C

Reference

1. Jakobsson, M. and Juels, A. Proofs of work and bread pudding protocols (extended abstract). In B. Preneel, Ed., *Secure Information Networks*. IFIP, The International Federation for Information Processing, vol. 23. Springer, Boston, MA, 1999.

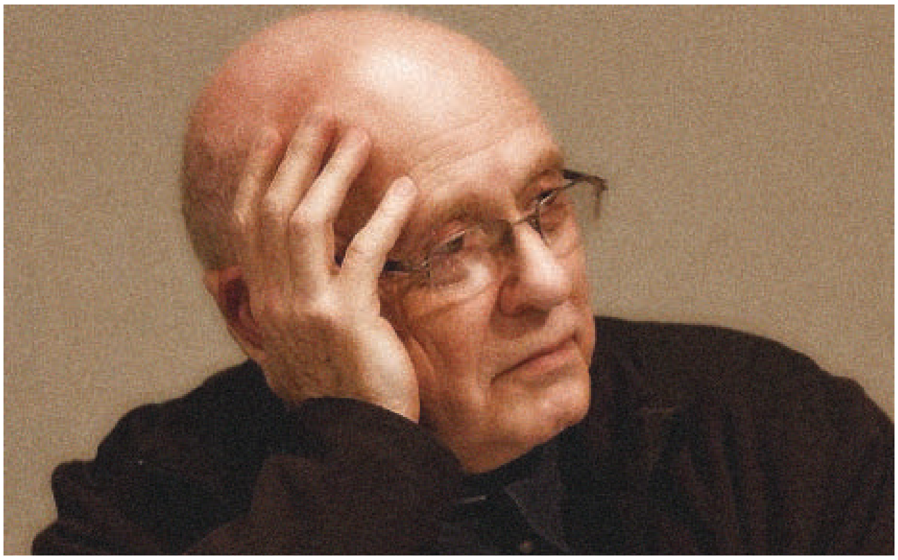
Nicholas Weaver (nweaver@icsi.berkeley.edu) is a researcher at the International Computer Science Institute and a lecturer in the CS department at UC Berkeley. He wishes to thank Steve Bellovin for his constructive shepherding of this column.

DOI:10.1145/3208097

Peter J. Denning

The Profession of IT
An Interview with Dave Parnas

A discussion of ideas about software engineering.



DAVID LORGE PARNAS is well known for his insights into how best to teach software engineering. Parnas has been studying software design and development since 1969, and has received more than 25 awards for his contributions. In 2007, he shared the IEEE Computer Society's 60th anniversary award with computer pioneer Maurice Wilkes. He received B.S., M.S., and Ph.D. degrees in Electrical Engineering from Carnegie Mellon University. He has published more than 285 papers, many of which are considered classics. He designed the CEAB (Canadian Engineering Accreditation Board) accredited McMaster University Software Engineering program, where he is now professor emeritus.

He and seven colleagues have articulated an approach based on actionable capabilities rather than concepts. *Communications* columnist Peter J. Denning had a conversation with Parnas about these ideas.

PETER J. DENNING: You have a long-standing interest in methods for producing reliable and safe software. In 1971, you first articulated the principle of information hiding.[2–4] Why is this still important?

DAVID LORGE PARNAS: The concept of information hiding is based on the observation that changes outside a program (such as a revision of other programs, hardware changes, or requirements changes) only affect the correctness of that program if information that would have to be used when showing the correctness of that program was invalidated by the change. Consequently, software should be organized so that dependence on information that is likely to change is restricted to a small, *clearly identified*, set of programs. Applying this principle results in programs that are easier to understand, easier to maintain, and less likely to contain errors.

As expressed in my early papers, the information hiding principle applies to all programs in any programming language. It serves as a guideline for software developers and maintainers. The first of the three papers introduced the basic idea; the second illustrated it for a simple system and the third showed the surprising implications of the principle when applied to more complex systems; that paper proposes a software structure that is very different from what is usually found in such systems. It also illustrates how that structure can be documented.

In 1968 and 1969, two famous NATO conferences discussed a new field—software engineering—to help us achieve reliable and trustworthy software systems. Why?

Most of those who attended the two conferences had been educated in either science or mathematics, but drifted into building programs for others to use (software). They realized that their education had taught them how to add to the world's knowledge, but not how to build products. Noting that traditional engineering education taught methods of designing and constructing reliable products, they proposed that that software development should be regarded and taught as an engineering discipline rather than as a branch of science.

How did universities respond to the conferences?

After the conferences, software engineering was treated as the area within computer science that studied ways to produce safe and reliable products. Some included project management within the field.

The initial response was to add a single one-semester course entitled "Software Engineering" to CS curriculums. It was not clear what should be taught in such a course. When I was assigned to teach that course, I asked, "What should it cover?" The only answer was "Dave, you are an engineer—you figure it out."

Later, as software became more important, CS departments defined software engineering "tracks" that included additional required courses—such as compilers, database systems, and software test methods. Important aspects of engineering such as interface design and fault tolerance were rarely included.

On a number of occasions, you wrote sharp critiques of many of these programmes.[a] What was the basis of your criticisms? Did they produce results?

My first critique [6] complained about many aspects of the "tracks." I pointed out they were teaching topics that interested the teachers rather than what the students would need to know as professional software developers.

My second major commentary[7] was written after my university (McMaster) had formed a new department and was offering a program designed to be taught in the engineering faculty rather than the science faculty. It had no courses in common with the previously existing CS program. Students took the same first-year courses as all other engineering students; only at the end of that common first year could they select software engineering as their programme.

The point of both of these papers was that software engineering education should be considered professional education (like architecture, medicine, law or engineering) rather than based on a "liberal arts" model like physics or mathematics.

Our programme was designed to be accredited by the Canadian Engineering Education Board. In the year we graduated our first students, two other Canadian Universities were also accredited and graduated students.

Both papers generated a lot of discussion. The programme described in the 1998 paper[7] was well received by students and employers. Unlike graduates of CS programmes, our SE graduates could be licensed as professional engineers after passing the usual law and ethics exams.

In 2017, you chaired a committee that wrote a report about software development programmes.[1] The resulting paper takes a capabilities-based approach to specifying the goals of professional programs in software development. How does this approach differ from the more common approaches?

Previous efforts to prescribe the content of CS and SE programs were based on the concept of a "Body of Knowledge." They specified what the students should know when they graduated.

Noting that these were professional programs, we chose to specify what the graduates should be able to do upon graduation. Our goal was to allow individual institutions to choose the knowledge and methods that would be taught provided that they gave the graduates the required capabilities. Those who read the 2017 paper will see that the approach is quite different from earlier approaches to curriculum specification. It emphasizes software development capabilities, not the name and content of the courses.

The 2017 paper stresses the difference between CS programmes and professional software development programmes, basing your approach on very old observations by Brian Randell (an author of the reports on the 1968 and 1969 conferences) and Fred Brooks, author of the very popular book *The Mythical Man Month*. What did they say that attracted you?

Randell described software engineering as "multiperson development of multiversion programs." Brooks said that, in addition to writing code, software engineering required both combining separately written programs and "productizing" them—that is, making them suitable for use by people who had not written them. These topics usually received little or no attention in traditional programming courses.

We were able to identify a set of capabilities required to do the things that Randell and Brooks had identified as differentiating software development from basic programming. The accompanying table lists those capabilities but readers really should read the paper to see what is meant. They will find detailed, concrete justifications, definitions, and guidelines.

List of capabilities.

- Communicate precisely between developers and stakeholders
- Communicate precisely among developers
- Design human-computer interfaces
- Design and maintain multiversion software
  - Identify and separate changeable concerns
  - Document to ease revision
  - Use parameterization
  - Design software that can be moved to many platforms
  - Design software that is easily extended or contracted
  - Design and maintain products that will be offered in many versions
- Design software for reuse
- Ensure software products meet quality standards
- Develop secure software
- Create and use models in system development
- Specify, predict, analyze, and evaluate performance
- Be disciplined in development and maintenance
- Use metrics in system development
- Manage complex projects

You used "software systems engineering" to denote the class of programmes you were discussing. That class included both specialized programmes and general software engineering programmes. You listed 12 capabilities that should be imparted to software systems engineers. They are all engineering capabilities. How should this be used?

[a] "Programme" is used in the context of this interview to refer to a course of study at a university and to differentiate from a software program.

The graduate capabilities list is intended as a checklist for those teaching software development. They should be asking, "Will our graduates have these capabilities?" The answer should be: "Yes, all of them." If not, the institutions should be redesigning their programmes so that they can answer "Yes!"

Do you advocate that SE and CS education would both be better if they were kept separate?

The two are as distinct as physics and mechanical engineering. The physics taught in both programmes would overlap but the engineers will be taught how to use the material to build reliable products while the physics majors are taught how to add to the body of knowledge that constitutes the science.

Professional programmes tend to be more tightly constrained than science programs because there are many things that a professional must know to be licensed and allowed to practice. A science student is often allowed to make more choices and become a specialist.

It is difficult (though not impossible) to have both types of programmes in one department.

You have said a professional software engineering programme would appeal to the students who want to learn how to build things for others to use. Are CS departments out of tune with most of their students?

The CS departments I have visited have a diverse set of students. Some want to be developers, while others want to be scientists. Many departments offer a compromise programme that is far from ideal for either group. That is why I prefer two distinct programmes taught by different (though not necessarily disjoint) sets of faculty members.

In 1985, you took a strong stand against the U.S. strategic defense initiative (SDI),[5] which promised to build an automated ballistic missile defense (BMD) system that would allow the U.S. to abandon its intercontinental ballistic missiles (ICBMs).You maintained the software could not be trusted enough for the U.S. to eliminate its missiles. We have BMD systems today, were you wrong?

Not at all! SDI was predicted by its advocates to be ready in six years and capable of intercepting and destroying sophisticated missiles including newer designs designed to defeat a BMD system by taking evasive measures. The system described by President Reagan would have been impossible to test under realistic conditions. The BMD systems in use today (33 years later) are not reliable even when facing unsophisticated rockets. No ICBM systems have been dismantled because BMD systems cannot be trusted.

Noting that these were professional programmes, we chose to specify what the graduates should be able to do upon graduation.

Do you see a relationship between the BMD claims made in the 1980s and today's claims about artificial intelligence?

Both fields are characterized by hyperbolic claims, overly optimistic predictions, and a lack of precise definitions. Both will produce systems that cannot be trusted.[9]

In 2007, you published a short paper[8] that criticized the evaluation of researchers by the number of papers they publish. What led you to publish such a paper?

I have served on many committees that evaluate faculty members for promotion and many others that evaluate research proposals. All too often, I have been disappointed to learn that most of my fellow committee members had not read any of the applicant's papers. They had merely counted the papers and (sometimes) estimated the selectivity of the journals and conferences. On two occasions colleagues complained when I started to discuss problems in the applicant's papers (which I had read). They said that the referees had already read the papers and approved them so I had no right to evaluate them. In effect, they said I was "out of line" in reading the papers and evaluating their contribution. The final straw came when someone published a computer program for doing the work of a committee by counting the papers and computing a score. The fact that such simple programs would often get the same result as the committee showed me that committee members were not doing their jobs. For example, referees of an individual paper cannot detect an author that publishes the same results several times using different titles and wording. We have scientists on the evaluation committees precisely because they have the expertise to read the papers and evaluate the contribution made by the author. If they don't do that, we don't need them. Sometimes a single paper is a far more important contribution than a dozen shallow or repetitive papers. Simply counting papers is not enough.

I have observed that people being evaluated for appointments or grants learn how to "play the game." If they see that they will be evaluated by people who won't read the papers but just count them, they know how to increase their score without actually improving the contribution. My 2007 paper discussed some techniques that researchers use to make themselves look better than they are. C

References

1. Landwehr, C., Ludewig, J., Meersman, R., Parnas, D.L., Shoval, P., Wand, Y., Weiss, D., and Weyuker, E. Software systems engineering programmes: A capability approach. *J. of Systems and Software 125*, (2017), 354–364.
2. Parnas, D.L. Information distributions aspects of design methodology. In *Proceedings of IFIP Congress '71*, Booklet TA-3, 1971, 26–30.
3. Parnas, D.L. On the criteria to be used in decomposing systems into modules. *Commun. ACM 15*, 12 (Dec. 1972), 1053–1058.
4. Parnas, D.L., Clements, P.C., and Weiss, D.M. The modular structure of complex systems. *IEEE Transactions on Software Engineering SE-11*, 3 (Mar. 1985), 259–266.
5. Parnas, D.L. Software aspects of strategic defense systems. *Commun. ACM 28*, 12 (Dec. 1985), 1326–1335.
6. Parnas, D.L. Education for computing professionals. *IEEE Computer 23*, 1 (Jan. 1990), 17–22.
7. Parnas, D.L. Software engineering programs are not computer science programs. *Annals of Software Engineering 6* (1998), 19–37. Reprinted in *IEEE Software* (Nov./Dec. 1999), 19–30.
8. Parnas, D.L. Stop the numbers game. *Commun. ACM 50*, 11 (Nov. 2007), 19–21.
9. Parnas, D.L. The real risks of artificial intelligence. *Commun. ACM 60*, 10 (Oct. 2017), 27–31.

Peter J. Denning (pjd@nps.edu) is Distinguished Professor of Computer Science and Director of the Cebrowski Institute for information innovation at the Naval Postgraduate School in Monterey, CA, is Editor of ACM *Ubiquity*, and is a past president of ACM.

The author's views expressed here are not necessarily those of his employer or the U.S. federal government.

DOI:10.1145/3208099

George V. Neville-Neil

Kode Vicious
Watchdogs vs. Snowflakes

Taking wild guesses.

Dear KV,
I have been working with a distributed job-control system for a large computing cluster for the past year. The system was developed in-house by one of the co-founders of the company, and he continues to work on it sporadically, while a small team of us adds new features and tries to fix bugs. The code isn't terrible, but it has one major defect—if the system doesn't have enough jobs in its queues, it tends to freeze up. I have been working with one other person on my team to diagnose the problem, but it has been assigned a very low priority by management because as long as we add dummy jobs when the system would otherwise be idle, the bug does not occur. I have never seen a system act like this, and I have to wonder: Is this kind of problem common in distributed job-control systems?

Jobless

IMAGE COLLAGE BY ANDRIJ BORYS ASSOCIATES/SHUTTERSTOCK

Dear Jobless,
Is the specific problem of a system freezing up because of starvation common in distributed job-control systems? It has been my experience that each distributed system is a precious snowflake—and KV does not like snow!

Let's first address the high-level issue—the fact that no one cares if you fix the bug, because if you put in dummy jobs, the system "just works." The phrase "just works" is one of the most overused in computing, and what it really indicates, in this case, is that someone is intellectually lazy, or that his or her motivation lies elsewhere. "Why should we care that we're running our systems at 100% power draw, when fixing the problem would cost time and money?" Apart from the fact that computing now consumes a significant percentage of the world's electricity, leaving a bug like this unaddressed can have other deleterious side effects.

That a system can randomly jam does not just indicate a serious bug in the system; it is also a major source of risk. You do not say what your distributed job-control system controls, but let's just say I hope it is not something with significant, real-world side effects—like a power station, jet aircraft, or financial trading system. The risk, of course, is that the system will jam, not when it is convenient for someone to add a dummy job to clear the jam, but during some operation that could cause data loss or return incorrect results. I rather suspect that having a system like this jam while coordinating, for example, the balancing of electrical power across a power grid would have spectacular and perhaps fatal results.

I am not saying every bug must be fixed at the expense of doing otherwise productive work, but it is bugs like this one that, in my experience, tend to hit at the absolute worst possible time. If the team knew about the bug in advance, it just leads to embarrassment when they must admit they knew about such a risk before it actually happened.

It is difficult to say much about the technical issue without looking into the system itself. (Remember KV's earlier comment about snowflakes.) The most common way of handling this type of freezing is itself not completely satisfying, and that is to have a watchdog process that sees if the system is making progress and restarts it after a suitable timeout when it believes the system is stuck.

There are several problems with the watchdog approach. The first is what the watchdog will actually do. Some watchdogs operate by restarting a stuck process, and they do this bluntly, by killing the process and restarting it. If the computations undertaken by the system are all idempotent, then there is little risk because any operation that did not complete will be restarted from the beginning and should have no side effects. Most systems have side effects, which means such restarts can cause a cascade of errors through the whole system. If the errors are obvious, then a human operator might be able to roll back the system to a good, known state and start the system again. But what if the errors are a type of silent corruption, returning incorrect answers (as I mentioned at the beginning of this column)? In that case, the watchdog is likely to do more harm than good.

Even if a watchdog approach is not otherwise harmful, there is a second problem of choosing an appropriate timeout duration. Since the system becomes jammed when it does not have enough work, some people will want to set the watchdog timer to be very fast so as to prevent these jams from reducing the overall efficiency of the system. A very short watchdog timeout has the potential to make the system thrash, since each restart caused by the watchdog firing will require the system to do work to return to its running state. All the work done by the system when a process is restarted is pure overhead; it does not help the system perform the work it was intended to do. Conversely, setting a watchdog timeout to be too long risks having the system remain stuck for long periods, again reducing overall efficiency. Too often, the choice of these timeouts is accomplished by a form of black magic, referred to as "taking a wild guess," followed by a heuristic, which is referred to as "taking another wild guess," to see if it is better than the first.

There are several problems with the watchdog approach. The first is what the watchdog will actually do.

Do not underestimate the number of production systems that use these approaches. I believe if we truly knew how many of the systems we depend on every day used black magic under the hood, we would all be more likely to buy land in Wyoming, build bunkers, and live in them.

Unfortunately, as KV has discussed before, debugging distributed systems is difficult, but it turns out that not debugging them and having them fail catastrophically makes for even more difficult days.

KV

Related articles on queue.acm.org

Poisonous Programmers
Kode Vicious
https://queue.acm.org/detail.cfm?id=1348585

From the EDVAC to WEBVACs
Daniel C. Wang
https://queue.acm.org/detail.cfm?id=2756508

Too Big NOT to Fail
Pat Helland, Simon Weaver, and Ed Harris
https://queue.acm.org/detail.cfm?id=3077383

George V. Neville-Neil (kv@acm.org) is the proprietor of Neville-Neil Consulting and co-chair of the ACM *Queue* editorial board. He works on networking and operating systems code for fun and profit, teaches courses on various programming-related subjects, and encourages your comments, quips, and code snips pertaining to his *Communications* column.

Calendar of Events

June 1–6
ISCA '18: The 45th Annual International Symposium on Computer Architecture, Los Angeles, CA, Contact: Timothy Pinakston, Email: tpink@usc.edu

June 2
FormaliSE '18: 6th Conference on Formal Methods in Software Engineering, Gothenburg, Sweden, Contact: Stefania Gnesi, Email: stefania.gnesi@isti.cnr.it

June 3–7
JCDL '18: The 18th ACM/IEEE Joint Conference on Digital Libraries, Fort Worth, TX, Contact: Jiangping Chen, Email: jiangping.chen@unt.edu

June 4–8
ASIA CCS '18: ACM Asia Conference on Computer and Communications Security, Incheon, Republic of Korea, Sponsored: ACM/SIG, Contact: Jong Kim, Email: jkim@postech.ac.kr

June 6–8
PerDis '18: The International Symposium on Pervasive Displays, Munich, Germany, Sponsored: ACM/SIG, Contact: Albrecht Schmidt, Email: albrecht.schmidt@gmail.com

June 9–10
DIS '18: Designing Interactive Systems Conference Workshops, Hong Kong, Sponsored: ACM/SIG, Contact: Ilpo Koskinen, Email: ilpo.koskinen@gmail.com

June 10–15
SIGMOD/PODS '18: International Conference on Management of Data, Houston, TX, Sponsored: ACM/SIG, Contact: Christopher Jermaine, Email: cmj4@rice.edu

June 11–13
DIS '18: Designing Interactive Systems Conference 2018, Hong Kong, Sponsored: ACM/SIG, Contact: Ilpo Koskinen, Email: ilpo.koskinen@gmail.com

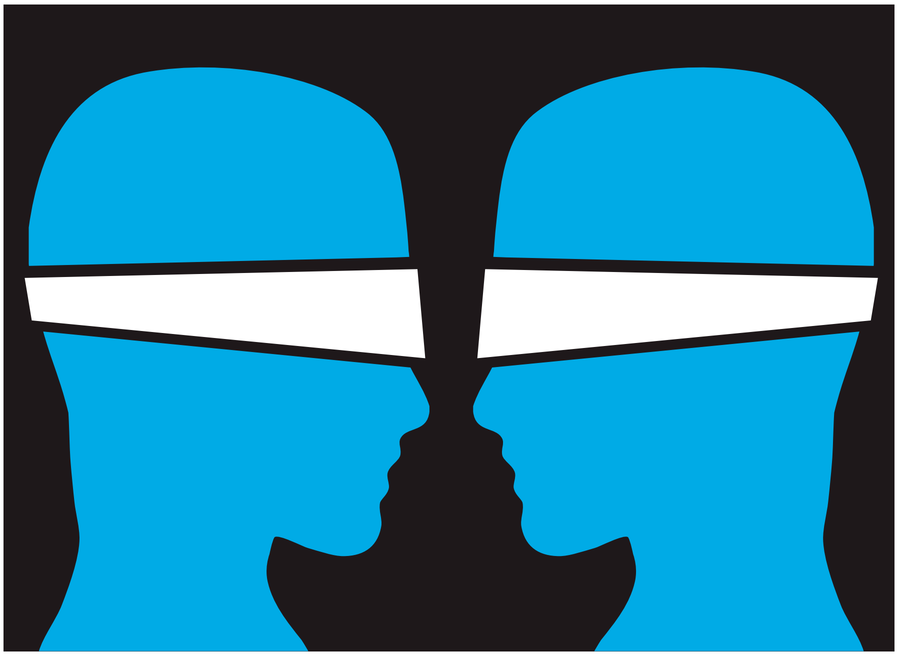
DOI:10.1145/3208157

C. Le Goues, Y. Brun, S. Apel, E. Berger, S. Khurshid, and Y. Smaragdakis

Viewpoint Effectiveness of Anonymization in Double-Blind Review

Assessing the effectiveness of anonymization in the review process.

PEER REVIEW IS a cornerstone of the academic publication process but can be subject to the flaws of the humans who perform it. Evidence suggests subconscious biases influence one's ability to objectively evaluate work: In a controlled experiment with two disjoint program committees, the ACM International Conference on Web Search and Data Mining (WSDM'17) found that reviewers with author information were 1.76x more likely to recommend acceptance of papers from famous authors, and 1.67x more likely to recommend acceptance of papers from top institutions.[6] A study of three years of the Evolution of Languages conference (2012, 2014, and 2016) found that, when reviewers knew author identities, review scores for papers with male-first authors were 19% higher, and for papers with female-first authors 4% lower.[4] In a medical discipline, U.S. reviewers were more likely to recommend acceptance of papers from U.S.-based institutions.[2]



These biases can affect anyone, regardless of the evaluator's race and gender.[3] Luckily, double-blind review can mitigate these effects[1,2,6] and reduce the perception of bias,[5] making it a constructive step toward a review system that objectively evaluates papers based strictly on the quality of the work.

Three conferences in software engineering and programming languages held in 2016—the IEEE/ACM International Conference on Automated Software Engineering (ASE), the ACM International Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA), and the ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)—collected data on anonymization effectiveness, which we[a] use to assess the degree to which reviewers were able to successfully deanonymize the papers' authors. We find that anonymization is imperfect but fairly effective: 70%–86% of the reviews were submitted with no author guesses, and 74%–90% of reviews were submitted with no correct guesses. Reviewers who believe themselves to be experts on a paper's topic were more likely to attempt to guess author identities but no more likely to guess correctly. Overall, we strongly support the continued use of double-blind review, finding the extra administrative effort minimal and well worth the benefits.

a Sven Apel and Sarfraz Khurshid were the ASE'16 PC chairs, Claire Le Goues and Yuriy Brun were the ASE'16 review process chairs, Yannis Smaragdakis was the OOPSLA'16 PC chair, and Emery Berger was the PLDI'16 PC chair.

IMAGERY BY LUCKY TEAM STUDIO

Methodology

The authors submitting to ASE 2016, OOPSLA 2016, and PLDI 2016 were instructed to omit author information from the author block and obscure, to the best of their ability, identifying information in the paper. PLDI authors were also instructed not to advertise their work. ASE desk-rejected submissions that listed author information on the first page, but not those that inadvertently revealed such information in the text. Authors of OOPSLA submissions who revealed author identities were instructed to remove the identities, which they did, and no paper was desk-rejected for this reason. PLDI desk-rejected submissions that revealed author identities in any way.

The review forms included optional questions about author identities, the answers to which were only accessible to the PC chairs. The questions asked if the reviewer thought he or she knew the identity of at least one author, and if so, to make a guess and to select what informed the guess. The data considered here refers to the first submitted version of each review. For ASE, author identities were revealed to reviewers immediately after submission of an initial review; for OOPSLA, ahead of the PC meeting; for PLDI, only for accepted papers, after all acceptance decisions were made.

Threats to validity. Reviewers were urged to provide a guess if they thought they knew an author. A lack of a guess could signify not following those instructions. However, this risk is small, for example, OOPSLA PC members were allowed to opt out uniformly and yet 83% of the PC members participated. Asking reviewers if they could guess author identities may have affected their behavior: they may not have thought about it had they not been asked. Data about reviewers' confidence in guesses may affect our conclusions. Reviewers could submit multiple guesses per paper and be considered correct if at least one guess matched, so making many uninformed guesses could be considered correct, but we did not observe this phenomenon. In a form of selection bias, all conferences' review processes were chaired by—and this Viewpoint is written by—researchers who support double-blind review.

Figure 1. Papers, reviews, reviewers, and author guesses. Reviewers include those on the program and external committees, but exclude chairs. All papers received at least three reviews; review load was non-uniform.

	ASE	OOPSLA	PLDI
Reviewers	79	37	111
Papers accepted	71	52	48
Papers rejected	263	144	240
Reviews	**1,029**	**636**	**1,154**
Did not contain a correct author guess	90.2%	74.4%	81.0%
Did not contain an author guess	86.4%	70.0%	74.3%
Tried to guess at least one author	14.7%	30.0%	25.7%
Guessed at least one author correctly	9.8%	25.6%	19.1%
All author guesses incorrect	3.8%	4.4%	6.7%
Reviews with a guess	**140**	**191**	**297**
Guess at least one author correctly	72.1%	85.3%	74.1%
Guess all authors incorrectly	27.9%	14.7%	25.9%
Papers reviewed	**334**	**196**	**288**
No one tried guessing authors	66.5%	41.8%	40.6%
Someone guessed an author correctly	24.6%	50.0%	44.1%
All guesses incorrect	9.0%	8.2%	15.3%

Figure 2. Guess rate, and correct guess rate, by self-reported reviewer expertise score (X: expert, Y: knowledgeable, Z: informed outsider).

	ASE		OOPSLA		PLDI	
	Guess	Correct	Guess	Correct	Guess	Correct
X	19.0%	74.7%	33.6%	86.7%	33.7%	74.2%
Y	11.2%	71.2%	29.3%	84.3%	24.6%	69.0%
Z	7.1%	55.6%	21.2%	83.3%	19.7%	48.6%

Anonymization Effectiveness

For the three conferences, 70%–86% of reviews were submitted without guesses, suggesting that reviewers typically did not believe they knew or were not concerned with who wrote most of the papers they reviewed. Figure 1 summarizes the number of reviewers, papers, and reviews processed by each conference, and the distributions of author identity guesses.

When reviewers did guess, they were more likely to be correct (ASE 72% of guesses were correct, OOPSLA 85%, and PLDI 74%). However, 75% of ASE, 50% of OOPSLA, and 44% of PLDI papers had no reviewers correctly guess even one author, and most reviews contained no correct guess (ASE 90%, OOPSLA 74%, PLDI 81%).

Are experts more likely to guess and guess correctly? All reviews included a self-reported assessment of reviewer expertise (X for expert, Y for knowledgeable, and Z for informed outsider). Figure 2 summarizes guess incidence and guess correctness by reviewer expertise. For each conference, X reviewers were statistically significantly more likely to guess than Y and Z reviewers ($p \leq 0.05$). But the differences in guess correctness were not significant, except the Z reviewers for PLDI were statistically significantly correct less often than the X and Y reviewers ($p \leq 0.05$). We conclude that reviewers who considered themselves experts were more likely to guess author identities, but were no more likely to guess correctly.

Are papers frequently poorly anonymized? One possible reason for deanonymization is poor anonymization. Poorly anonymized papers may have more reviewers guess, and also a higher correct guess rate. Figure 3 shows the distribution of papers by the number of reviewers who attempted to guess the authors. The largest proportion of papers (26%–30%) had only a single reviewer attempt to guess. Fewer papers had more guesses. The bar shading indicates the fractions of the author identity guesses that are correct; papers with more guesses have lower rates of incorrect guesses. Combining the three conferences' data, the χ^2 statistic indicates that the rates of correct guessing for papers with one, two, and three or more guesses are statistically significantly different ($p \leq 0.05$). This comparison is also statistically significant for OOPSLA alone, but not for ASE and PLDI. Comparing guess rates (we use one-tailed z tests for all population proportion comparisons) between paper groups directly: For OOPSLA, the rate of correct guessing is statistically significantly different between one-guess papers and each of the other two paper groups. For PLDI, the same is true between one-guess and three-plus-guess paper groups. This evidence suggests a minority of papers may be easy to unblind. For ASE, only 1.5% of the papers had three or more guesses, while for PLDI, 13% did. However, for PLDI, 40% of all the guesses corresponded to those 13% of the papers, so improving the anonymization of a relatively small number of papers would potentially significantly reduce the number of guesses. Since the three conferences only began using the double-blind review process recently, the occurrences of insufficient anonymization are likely to decrease as authors gain more experience with anonymizing submissions, further increasing double-blind effectiveness.

Are papers with guessed authors more likely to be accepted? We investigated if paper acceptance correlated with either the reviewers' guesses or with correct guesses. Figure 4 shows the acceptance rate for each conference for papers without guesses, with at least one correct guess, and with all incorrect guesses. We observed different behavior at the three conferences: ASE submissions were accepted at statistically the same rate regardless of reviewer guessing behavior. Additional data available for ASE shows that for each review's paper rating (strong accept, weak accept, weak reject, strong reject), there were no statistically significant differences in acceptance rates for submissions with different guessing behavior. OOPSLA and PLDI submissions with no guesses were less likely to be accepted ($p \leq 0.05$) than those with at least one correct guess. PLDI submissions with no guesses were also less likely to be accepted ($p \leq 0.05$) than submissions with all incorrect guesses (for OOPSLA, for the same test, $p = 0.57$). One possible explanation is that OOPSLA and PLDI reviewers were more likely to affiliate work they perceived as of higher quality with known researchers, and thus more willing to guess the authors of submissions they wanted to accept.

How do reviewers deanonymize? OOPSLA and PLDI reviewers were asked if the use of citations revealed the authors. Of the reviews with guesses, 37% (11% of all reviews) and 44% (11% of all reviews) said they did, respectively. The ASE reviewers were asked what informed their guesses. The answers were guessing based on paper topic (75 responses); obvious unblinding via reference to previous work, dataset, or source code (31); having previously reviewed or read a draft (21); or having seen a talk (3). The results suggest that some deanonymization may be unavoidable. Some reviewers discovered GitHub repositories or project websites while searching for related work to inform their reviews. Some submissions represented clear extensions of or indicated close familiarity with the authors' prior work. However, there also exist straightforward opportunities to improve anonymization. For example, community familiarity with anonymization, consistent norms, and clear guidelines could address the incidence of direct unblinding. However, multiple times at the PC meetings, the PC chairs heard a PC member remark about having been sure another PC member was a paper author, but being wrong. Reviewers may be overconfident, and sometimes wrong, when they think they know an author through indirect unblinding.

Figure 3. Distributions of papers by number of guesses. The bar shading indicates the fraction of the guesses that are correct.

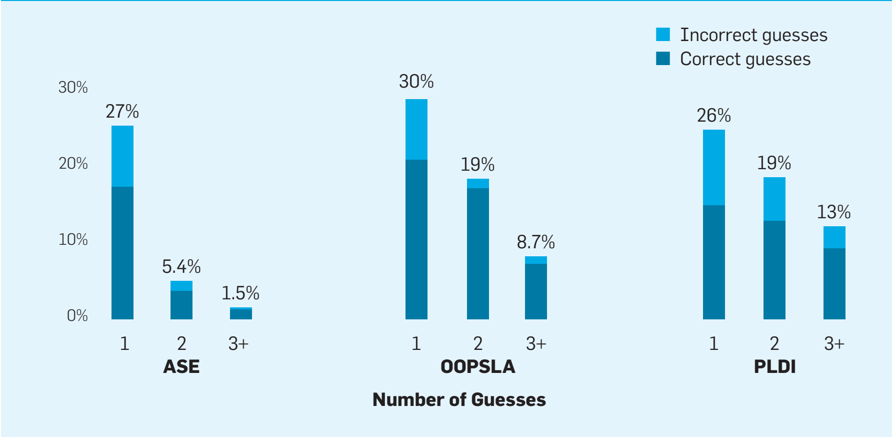


Figure 4. Acceptance rate of papers by reviewer guessing behavior.

Papers with	ASE	OOPSLA	PLDI
No guesses	21.2%	20.7%	6.8%
At least one correct guess	22.0%	31.6%	22.3%
All guesses incorrect	23.0%	25.0%	25.0%
All papers	21.3%	26.5%	16.7%

PC Chairs' Observations

After completing the process, the PC chairs of all three conferences reflected on the successes and challenges of double-blind review. All PC chairs were strongly supportive of continuing to use double-blind review in the future. All felt that double-blind review mitigated effects of (subconscious) bias, which is the primary goal of using double-blind review. Some PC members also felt so, indicating anecdotally that they were more confident their reviews and decisions had less bias. One PC member remarked that double-blind review is liberating, since it allows for evaluation without concern about the impact on the careers of people they know personally.

All PC chairs have arguments in support of their respective decisions on the timing of revealing the authors (that is, after review submission, before PC meeting, or only for accepted papers). The PLDI PC chair advocated strongly for full double-blind, which enables rejected papers to be anonymously resubmitted to other double-blind venues with common reviewers, addressing one cause of deanonymization. The ASE PC chairs observed that in a couple of cases, revealing author identities helped to better understand a paper's contribution and value. The PLDI PC chair revealed author identities on request, when deemed absolutely necessary to assess the paper. This happened extremely rarely, and could provide the benefit observed by the ASE PC chairs without sacrificing other benefits. That said, one PC member remarked that one benefit of serving on a PC is learning who is working on what; full anonymization eliminates learning the who, though still allows learning the what.

Overall, none of the PC chairs felt the extra administrative burden imposed by double-blind review was large. The ASE PC chairs recruited two review process chairs to assist, and all felt the effort required was reasonable. The OOPSLA PC chair noted the level of effort required to implement double-blind review, including the management of conflicts of interest, was not high. He observed that it was critical to provide clear guidance to the authors on how to anonymize papers (for example, http://2016.splashcon.org/track/splash-2016-oopsla#FAQ-on-Double-Blind-Reviewing). PLDI allowed authors to either anonymize artifacts (such as source code) or to submit non-anonymized versions to the PC chair, who distributed to reviewers when appropriate, on demand. The PC chair reported this presented only a trivial additional administrative burden.

All PC chairs were strongly supportive of continuing to use double-blind review in the future.

The primary source of additional administration in double-blind review is conflict of interest management. This task is simplified by conference management software that straightforwardly allows authors and reviewers to declare conflicts based on names and affiliations, and chairs to quickly cross-check declared conflicts. ASE PC chairs worked with the CyberChairPro maintainer to support this task. Neither ASE nor OOPSLA observed unanticipated conflicts discovered when author identities were revealed. The PLDI PC chair managed conflicts of interest more creatively, creating a script that validated author-declared conflicts by emailing PC members lists of potentially conflicted authors mixed with a random selection of other authors, and asking the PC member to identify conflicts. The PC chair examined asymmetrically declared conflicts and contacted authors regarding their reasoning. This identified erroneous conflicts in rare instances. None of the PC chairs found identifying conflicts overly burdensome. The PLDI PC chair reiterated that the burden of full double-blind reviewing is well worth maintaining the process integrity throughout the entire process, and for future resubmissions.

Conclusion

Data from ASE 2016, OOPSLA 2016, and PLDI 2016 suggest that, while anonymization is imperfect, it is fairly effective. The PC chairs of all three conferences strongly support the continued use of double-blind review, find it effective at mitigating (both conscious and subconscious) bias in reviewing, and judge the extra administrative burden to be relatively minor and well worth the benefits. Technological advances and the now-developed author instructions reduce the burden. Having a dedicated organizational position to support double-blind review can also help. The ASE and OOPSLA PC chairs point out some benefits of revealing author identities midprocess, while the PLDI PC chair argues some of those benefits can be preserved in a full double-blind review process that only reveals the author identities of accepted papers, while providing significant additional benefits, such as mitigating bias throughout the entire process and preserving author anonymity for rejected paper resubmissions. C

References

1. Budden, et al. Double-blind review favours increased representation of female authors. *Trends in Ecology and Evolution 23*, 1 (Jan. 2008), 4–6.
2. Gastroenterology, Bethesda, MD, USA. U.S. and non-U.S. submissions: An analysis of reviewer bias. *JAMA 280*, 3 (July 1998), 246–247.
3. Moss-Racusin, C.A. et al. Science faculty's subtle gender biases favor male students. *PNAS 109*, 41 (Apr. 2014), 16474–16479.
4. Roberts, S.G. and Verhoef, T. Double-blind reviewing at EvoLang 11 reveals gender bias. *J. of Language Evolution 1*, 2 (Feb. 2016), 163–167.
5. Snodgrass, R. Single-versus double-blind reviewing: An analysis of the literature. *SIGMOD Record 35*, 3 (May 2006), 8–21.
6. Tomkins, A., Zhang, M., and Heavlin, W.D. Single versus double-blind reviewing at WSDM 2017. CoRR, abs/1702.00502, 2017.

Claire Le Goues (clegoues@cs.cmu.edu) is an Assistant Professor in the School of Computer Science at Carnegie Mellon University, Pittsburgh, PA, USA.

Yuriy Brun (brun@cs.umass.edu) is an Associate Professor in the College of Information and Computer Sciences at the University of Massachusetts Amherst, Amherst, MA, USA.

Sven Apel (apel@uni-passau.de) is a Professor of Computer Science and Chair of Software Engineering in the Department of Informatics and Mathematics at the University of Passau.

Emery Berger (emery@cs.umass.edu) is a Professor in the College of Information and Computer Sciences at the University of Massachusetts Amherst, Amherst, MA, USA.

Sarfraz Khurshid (khurshid@ece.utexas.edu) is a Professor in Electrical and Computer Engineering at the University of Texas at Austin, Austin, TX, USA.

Yannis Smaragdakis (smaragd@di.uoa.gr) is a Professor in the Department of Informatics at the University of Athens.

Kathryn McKinley suggested an early draft of the reviewer questions used by OOPSLA and PLDI. This work is supported in part by the National Science Foundation under grants CCF-1319688, CCF-1453474, CCF-1563797, CCF-1564162, and CNS-1239498, and by the German Research Foundation (AP 206/6).

DOI:10.1145/3190564

Article development led by acmqueue
queue.acm.org

Embracing failures for improving availability.

BY DIPTANU GON CHOUDHURY AND TIMOTHY PERRETT

Designing Cluster Schedulers for Internet-Scale Services

IN THE PAST decade, cluster schedulers have become the foundation of Internet-scale services that span hundreds or thousands of machines across many geographically distributed datacenters. Cluster schedulers have their origins in high-performance computing or managing services that run on warehouse-scale computing systems, which allow for efficient usage of computational resources. Schedulers also provide an API for application developers writing distributed applications such as Spark and MapReduce or building application management platforms such as DC/OS (Data Operating System). In the past few years open source schedulers such as Kubernetes, HashiCorp's Nomad, and Apache Mesos have democratized scale, allowing many enterprises to adopt scheduler technologies that were previously accessible only to companies like Facebook, Google, and Twitter.

Despite this apparent ubiquity, operating and implementing scheduling software is an exceedingly tricky task with many nuanced edge cases. This article highlights some of these cases based on the real-world experience of the authors designing, building, and operating a variety of schedulers for large Internet companies.

On a long enough timeline, everything will fail. Every day, many of the Internet services the world has come to rely upon have many small—even imperceptible—failures. Machines crash, APIs become intermittently latent, hard drives fail, and networks become saturated, but usually services powered by them don't fail to serve user requests. Though it may seem that recovering automatically from such failures is a solved problem, the reality is that many different processes are involved in orchestrating this recovery. Scheduling software is often the foundational infrastructure that allows a service to recover by interacting with various other datacenter services.

Modern distributed systems such as search, social networks, and cloud object stores consume many more resources than a handful of servers or even mainframes can provide. Such systems consume resources on the order of tens of thousands of machines, potentially spread across many datacenters. This often means it is not possible to treat the datacenter as a collection of servers, and the notion of an individual server is simply less relevant.

Software developers care only about available resources such as RAM, CPUs, and available bandwidth for accessing networked systems. To an extent, schedulers allow operators to ignore the distribution of compute resources. Within these bounds, when a workload running on a scheduler fails, the scheduler can simply look for resources

IMAGE BY ANDRIJ BORYS ASSOCIATES/SHUTTERSTOCK

within the available pool and reschedule that job for processing within the constraints imposed by the user. It is exactly at this moment of failure where things become interesting. Why did the workload fail? Was there an application problem? Or a machine-specific problem? Or was there perhaps a clusterwide or otherwise environmental problem? More importantly, how does the architecture of a scheduler impact the ability and timeline of a workload to recover? The answers to these questions directly affect and dictate how effective a scheduler can be in recovering that failed workload.

One of the responsibilities of a cluster scheduler is to supervise an individual unit of work, and the most primitive form of remediation is to move that workload to a different, healthy node; doing so will frequently solve a given failure scenario. When using any kind of shared infrastructure, however, you must carefully evaluate the bulkheading options applied for that shared infrastructure, and objectively assess the opportunity for cascading failure. For example, if an I/O-intensive job is relocated to a node that already hosts another I/O-intensive job, they could potentially saturate the network links in the absence of any bulkheading of IOPs (I/O operations), resulting in a degraded QoS (quality of service) of other tenants on the node.

In this article we highlight the various failure domains within scheduling systems and touch upon some of the practical problems operators encounter with machine, scheduler, environmental, and clusterwide failures. In addition, we provide some answers to dealing with the failures.

Considerations for Machine Failures

Failures at the machine level are probably the most common. They have a variety of causes: hardware failures such as disks crashing; faults in network interfaces; software failures such as excessive logging and monitoring; and problems with containerizer daemons.

All these failures result in degraded application performance or a potential incoherent cluster state. While all possible system failure modes are too numerous to mention in one article, within the realm of scheduling there are a handful of important factors to consider. Here, we cover details about the mechanics of the modern Linux operating system and how to mitigate the effects of typical failure modes encountered in the field.

Capacity planning. Regardless of where compute capacity is housed—public cloud or private datacenter—at some point capacity planning will be necessary to figure out how many machines are needed. Traditional methods[8] for capacity planning make assumptions about compute resources being entirely dedicated, where a given machine has a sole tenant. While this is a common industry practice, it is often ineffective as application authors tend to be overly optimistic about their runtime performance (resulting in insufficient capacity and potential outage at runtime) or overly cautious about resource consumption, leading to a high cost of ownership with a large amount of waste[5] when operating at scale.

Assuming an application has been benchmarked for performance and resource consumption, running that application on a scheduling system introduces additional challenges for capacity planning: how will shared components handle multitenancy? Common configurations have per-node utilities for routing traffic, monitoring, and logging (among other tasks); will these potentially impact those lab performance numbers? The probability is very high that they will have a (negative) impact.

Ensure the capacity plan includes headroom for the operating system, file systems, logging agents, and anything else that will run as a shared component. Critically, anything that is a shared component should have well-defined limits (where possible) on compute resources. Not provisioning an adequate amount of resources for system services inadvertently surfaces as busy-neighbor problems. Many schedulers allow operators to reserve resources for running system components, and correctly configuring these resource reservations can dramatically improve the predictability of application performance.

▸ *File systems.* Learn about the overhead and resource usage of file systems. This is useful, for example, when using ZFS to limit the ARC (adaptive replacement cache) to an acceptable size, or when planning to turn on deduplication or compression to account for the CPU cycles that ZFS itself is going to use. Consider another example: two containers doing a lot of file-system I/O with a very limited cache would end up invalidating each other's caches, resulting in poor I/O performance. Limiting file-system IOPs is not straightforward in Linux, since the block I/O and memory controller cannot interact with each other to limit the writeback I/O with traditional cgroup v1. The next version of cgroup *can* properly limit I/O, but a few controllers—such as the CPU controller—have not yet been merged.

▸ *Sidecars.* Logging, monitoring, or service meshes such as Envoy[2] can potentially use a considerable amount of resources, and this needs to be accounted for. For example, if a logging agent such as Fluentd[3] is forwarding logs to a remote sink, then the network bandwidth for that process should be limited so that containers can get their expected share of network resources for application traffic. Fair sharing of such resources is difficult, and therefore it is sometimes easier to run sidecars for every allocation on a node rather than sharing them, so that their resources can be accounted for under the cgroup hierarchy of the allocation.

▸ *Administration.* Policies for system or component configurations—such as garbage collection—should be based on the units that the underlying resource understands. For example, log retention policies based on a number of days are not effective on a node where the storage is limited by number of bytes—rotating logs every three days is useless if the available bytes are consumed within a matter of hours. Systems administrators often apply the same types of policies that they write for clusterwide log aggregation services for local nodes. This can have disastrous consequences at the cluster level where services are designed to scale out horizontally, where a workload might be spread across many nodes that have the same—or similar—hardware configuration.

These are some of the key elements to consider for capacity-planning purposes, but this is by no means an exhaustive set. Be sure to consider any environment-specific capacity bounds that might apply to your case, always basing your plan on real data about actual usage collected in the field.

The OOM (out of memory) **killer** in Linux steps in under extremely low memory conditions and kills processes to recover memory based on a set of rules and heuristics. The decisions made by the OOM killer are based on a so-called *oom_score*, which changes over time, based on certain rules, and is not deterministic in most situations.

The OOM killer is an important system component to keep in mind while designing schedulers that allow for oversubscription,[1] since they allow more tasks on a machine than actual resources. In such situations, it is important to design a good QoS module that actively tracks the resource usage of tasks and kills them proactively. If tasks consume more memory than they are allocated, the scheduler should kill the tasks before the overall resource utilization forces invocation of the system OOM killer. For example, QoS modules could implement their own strategy for releasing memory by listening for kernel notifications indicating memory pressure, and subsequently killing lower-priority tasks, which would prevent the kernel from invoking the OOM killer.

Having scheduler agents killing tasks allows for deterministic behavior and is easier to debug and troubleshoot. For example, in the Mesos cluster manager the Mesos agent runs a QoS controller that continuously monitors tasks that run with revocable resources and kills them if they interfere with normal tasks.

Leaking container resources. Since its introduction to the Linux kernel a decade ago, container technology has improved immensely. It is, however, still an imperfect world, and tools that have been built atop these foundations have added more complexity over time, opening the door to interesting and tricky-to-solve problems. One of the common runtime issues operators will encounter is the leaking of associated resources. For example, say you boot a container with a bridged networking mode; under the hood a virtual Ethernet adapter will be created. If the application crashes unexpectedly—and is not killed by an external agent—the container daemon can potentially leak virtual interfaces over time, which eventually causes a system problem when a moderate number of interfaces have been leaked. This causes new applications attempting to boot on that machine to fail, as they are unable to create virtual network adapters.

Remediating these types of failures can be difficult; the issue must first be monitored to keep track of the resources being created and garbage collected over time, ensuring that the leaking is either kept to a minimum or effectively mitigated. Operators often find themselves writing agents to disable scheduling on a node until resources become available to make sure a node is not running under pressure, or preemptively redistributing work before the issue manifests itself by causing an outage. It is best to surface such problems to the operators even if automated mitigation procedures are in place, since the problems are usually a result of bugs in underlying container runtimes.

Undersubscription of attached resources. Schedulers usually choose placements or bin-packing strategies for tasks based on node resources such as CPU, memory, disk, and capacity of the I/O subsystems. It is important, however, to consider the shared resources attached to a node, such as network storage or aggregated link layer bandwidth attached to the top of rack (ToR) switch to ensure such resources are allocated to a reasonable limit or are judiciously oversubscribed. Naive scheduler policies might undersubscribe node-local resource usage but oversubscribe aggregate resources such as bandwidth. In such situations, optimizing for cluster-level efficiency is better than local optimization strategies such as bin packing.

Multitenancy is one of the most difficult challenges for performance engineers to solve in an elastic, shared infrastructure. A cluster that is shared by many different services with varying resource usage patterns often shows so-called busy-neighbor problems. The performance of a service can become degraded because of the presence of other cotenant services. For example, on the Linux operating system, imposing QoS for the network can be complicated, so operators sometimes do not go through the effort of imposing traffic-shaping mechanisms for controlling throughput and bandwidth of network I/O in containers. If two network I/O-intensive applications run on the same node, they will adversely affect each other's performance.

Regardless of where compute capacity is housed—public cloud or private datacenter—at some point capacity planning will be necessary to figure out how many machines are needed.

Other common problems with multitenancy include cgroup controllers not accounting for certain resources correctly, such as the VFS IOP, where services that are very disk I/O-intensive will have degraded performance when colocated with similar services. Work has been ongoing in this area for the past five to six years to design new cgroup controllers[9] on Linux that do better accounting, but not all these controllers have yet been put into production. When workloads use SIMD (single instruction multiple data) instructions such as those from Intel's AVX-512 instruction set, processors throttle the CPU clock speed to reduce power consumption, thereby slowing other workloads running on the same CPU cores that are running non-SIMD instructions.[6]

Fair sharing of resources is often the most common approach offered by schedulers, and shares of resources are often expressed via scalar values. Scalar values are easier to comprehend from an end-user perspective, but in practice they do not always work well because of interference.[7] For example, if 100 units of IOPs are allocated to two workloads running on the same machine, the one doing sequential I/O may get a lot more throughput than the one performing random I/O.

Considerations for Cluster-Level Failures

Most of the failures that wake up operators in the middle of the night have affected entire clusters or racks of servers in a fleet. Cluster-level failures are usually triggered because of bad configuration changes, bad software deployment, or in some cases because of cascading failures in certain services that result in resource contention in a multitenant environment. Most schedulers come with remediation steps for unhealthy tasks such as restarts or eviction of lower-priority tasks from a node to reduce resource contention. Clusterwide failures indicate a problem far bigger than local node-related problems that can be solved with local remediation techniques.

Such failures usually require paging on-call engineers for remediation actions; however, the scheduler can also play a role in remediation during such failures. The authors have written and deployed schedulers that have clusterwide failure detectors and would prevent nodes from continuously restarting tasks locally. They also allow operators to define remediation strategies, such as reverting to a last known good version or decreasing the frequency of restarts, stopping the eviction of other tasks, among others, before the operators can debug possible causes of failure. Such failure-detection algorithms usually take into consideration the health of tasks cotenant on the same machine to differentiate service-level failures from other forms of infrastructure-related failures.

Clusterwide failures should be taken seriously by scheduler developers; the authors have encountered failures that have generated so many cluster events that they saturated the scheduler's ability to react to failures. Therefore, sophisticated measures must be taken to ensure the events are sampled without losing the context of the nature and magnitude of the underlying issues. Depending on the magnitude of failure, the saturation of events often brings operations to a standstill unless it is quickly mitigated. Here, we cover some of the most frequently used techniques for mitigating cluster-level failures.

Bad software push. Most cluster-level job failures are the result of bad software pushes or configuration changes. It can often be useful to track the start time of such failures and correlate them with cluster events such as job submissions, updates, and configuration changes. Another common, yet simple, technique for reducing the likelihood of cluster-wide failures in the face of bad software pushes is a rolling update strategy that incrementally deploys new versions of software only after the new instances have proven to be healthy or working as expected from the perspective of key metrics. Schedulers such as Nomad and Kubernetes come with such provisions. They move on to deploying newer versions of software only when the current set of tasks passes health checks and stops deployments if they start encountering failures.

Most of the failures that wake up operators in the middle of the night have affected entire clusters or racks of servers in a fleet.

System software failure. System software, such as the Docker daemon and monitoring and logging software, is an integral part of the scheduler infrastructure and thus contributes to the health of the cluster. New versions of such software have often been deployed only to find that they cause failures after some period of time in a cluster. In one instance a specific Auto Scaling Group on AWS (Amazon Web Services) started misbehaving a few days after the cluster joined the scheduler infrastructure; it turned out that a new version of Docker had been rolled out, which had functional regressions.

In most cases, the best strategy for dealing with such failures is to disable scheduling on those machines and drain the assigned work to force the relocation of the workload to elsewhere in the datacenter. Alternatively, you could introduce additional resource capacity with a working configuration of all system software, such that pending workloads could be scheduled successfully.

Such failures affect the tasks of all jobs in a specific cluster or resource pool; hence, schedulers should have good mechanisms for dealing with them. A robust scheduler design should ideally be able to detect an issue with a given cluster or resource pool and proactively stop scheduling workloads there. This kind of proactive remediation event should be included in the telemetry information being emitted by the scheduler so that on-call engineers can further debug and resolve the specific problem.

Shared resources failures. Modes of failure at the infrastructure level include fiber cuts; faulty power distribution for a machine, rack, or ToR switch; and many other environmental possibilities. In such cases, other than moving affected workloads to unaffected systems, a scheduler can do very little to mitigate the problem.

In some cluster schedulers, the default behavior when nodes become disconnected from the network is to begin killing tasks. Operationally, this can cause significant challenges when nodes return to a healthy state. In most cases, it is preferable to delegate the complexity of guaranteeing a fixed number of currently operational tasks to the application itself. This typically makes the scheduling system easier to operate and allows the application to get precisely the consistency and failure semantics it desires. Tasks should be allowed to join the cluster gracefully when the failures are mitigated. Such measures decrease the churn in the cluster and allow for its faster recovery.

Depletion of global resources. In addition to node resources, global resources such as aggregate bandwidth or power usage within the cluster should be tracked by the scheduler's resource allocator. Failure to track these global resources could result in placement decisions oversubscribing cluster resources, causing bottlenecks that create hotspots within the datacenter, thereby reducing the efficiency of the provisioned hardware.

For example, bin packing too many network I/O-intensive tasks in a single rack might saturate the links to the datacenter's backbone, creating contention, even though network links at the node level might not be saturated. Bin packing workloads very tightly in a specific part of the datacenter can also have interesting or unexpected side effects with respect to power consumption, thereby impacting the available cooling solutions.

Software-distribution mechanisms. It is very important to understand the bottlenecks of software-distribution mechanisms. For example, if the aggregate capacity of a distribution mechanism is 5Gbps, launching a job with tens of thousands of tasks could easily saturate the limit of the distribution mechanism or even of the shared backbone. This could have detrimental effects on the entire cluster and/or the running services. Parallel deployments of other services can often be affected by such a failure mode; hence, the parallelism of task launches must be capped to ensure no additional bottlenecks are created when tasks are deployed or updated.

Keep in mind that distribution mechanisms that are centralized in nature, such as the Docker registry, are part of the availability equation. When these centralized systems fail, job submission or update requests fail as well, thereby putting services at risk of becoming unavailable if they, too, are updated. Extensive caching of artifacts on local nodes to reduce pressure on centralized distribution mechanisms can be an effective mitigation strategy against centralized distribution outages. In some instances, peer-to-peer distribution technologies such as BitTorrent can further increase the availability and robustness of such systems.

Back-off strategies for remediation actions. Certain workloads might not perform well on any node in the cluster and might be adversely affecting the health of the services and the nodes. In such cases, the schedulers must detect the trend while reallocating workloads or bring additional capacity to ensure they do not deplete global resources, such as API call limits of cloud providers, or adversely affect cotenant workload, thereby causing cascading failures.

Control-Plane Failure Considerations

Control planes within schedulers have a different set of failure considerations than compute nodes and clusters, as the control plane must react to changes in the cluster as they happen, including various forms of failure. Software engineers writing such systems should understand the user interaction, scale, and SLA (service-level agreement) for workloads and then derive an appropriate design that encompasses handling failures in the control plane. Here, we look at some of the important design considerations for control-plane developers.

Reliable cluster state reconciliation. At the end of the day, most schedulers are just managing cluster state, supervising tasks running on the cluster, and ensuring QoS for them. Schedulers usually track the cluster state and maintain an internal finite-state machine for all the cluster objects they manage, such as clusters, nodes, jobs, and tasks. The two main ways of cluster state reconciliation are level- and edge-triggered mechanisms. The former is employed by schedulers such as Kubernetes, which periodically looks for unplaced work and tries to schedule that work. These kinds of schedulers often suffer from having a fixed baseline latency for reconciliation.

Edge-triggered scheduling is more common. Most schedulers, such as Mesos and Nomad, work on this model. Events are generated when something changes in the cluster infrastructure, such as a task failing, node failing, or node joining. Schedulers must react to these events, updating the finite state machine of the cluster objects and modifying the cluster state accordingly. For example, when a task fails in Mesos, the framework gets a TASK_LOST message from the master and reacts to that event based on certain rules, such as restarting the task elsewhere on the cluster or marking a job as dead or complete. Nomad is similar: it invokes a scheduler based on the type of the allocation that died, and the scheduler then decides whether the allocation needs to be replaced.

While event-driven schedulers are faster and more responsive in practice, guaranteeing correctness can be harder since the schedulers have no room to drop or miss the processing of an event. Dropping cluster events will result in the cluster not converging to the right state; jobs might not be in their expected state or have the right number of tasks running. Schedulers usually deal with such problems by making the agents or the source of the cluster event resend the event until they get an acknowledgment from the consumer that the events have persisted.

Quotas for schedulers. Schedulers are usually offered to various teams in an organization for consuming compute infrastructure in the datacenter. Schedulers usually implement quotas, which ensure that various jobs have the right amount of resources on the clusters during resource contention. Besides quotas for compute resources on the compute clusters, scheduler developers also must consider how much time schedulers spend doing scheduling per job. For example, the amount of time it would take to schedule a batch job with 15,000 tasks would be much more than for a job with 10 tasks. Alternatively, a job might have a few tasks but very rigorous placement constraints. The scheduler might spend varying amounts of time serving various jobs from various teams based on constraints and volume of the tasks or churn in the cluster. Monolithic schedulers, which centralize all the scheduling work, are more prone to these kinds of problems than two-level schedulers such as Mesos, where operators can run multiple frameworks to ensure various schedulers are serving a single purpose and thereby not sharing scheduling time for anything else.

With monolithic schedulers it is important to develop concepts such as quotas for various types of jobs or teams. Another possibility for scaling schedulers is to do parallel scheduling in a similar manner to Nomad, where the operators can run many scheduler instances that work in parallel and can decide how many scheduler processes they want to run for a certain job type.

Recovering cluster state from failures. Scheduler operators want AP (CAP available, partition tolerant) systems in practice because they prefer availability and operability over consistency. The convergence of the cluster state eventually must be guaranteed after all the cluster events have been processed or by some form of reconciliation mechanism. Most real-world schedulers, however, are built on top of highly consistent coordination systems such as ZooKeeper or etcd, because building and reasoning about such distributed systems are easier when the data store provides guarantees of linearizability. It is not unheard of for schedulers to lose their entire database for a few hours. One such instance was when AWS had a Dynamo outage, and a large scheduler operating on top of AWS was using Dynamo to store cluster state. There is not a lot that can be done in such situations, but scheduler developers have to consider this scenario and develop with the goal of causing the least impact to running services on the cluster.

Some schedulers such as Mesos allow operators to configure a duration after which an agent that is disconnected from the scheduler starts killing the tasks running on a machine. Usually this is done with the assumption that the scheduler is disconnected from the nodes because of failures such as network partitions; since the scheduler thinks the node is offline, it has already restarted the tasks on that machine somewhere else in the cluster. This does not work when schedulers are experiencing outages or have failed in an unrecoverable manner. It is better to design scheduler agents that do not kill tasks when the agent disconnects from the scheduler, but instead allow the tasks to run and even restart them if a long-running service fails. Once the agents rejoin the cluster, the reconciliation mechanisms should converge the state of the cluster to an expected state.

The process of restoring a cluster's state when a scheduler loses all its data is complicated, and the design depends largely on the architecture and data model of the scheduler. On Apache Mesos, the scheduler frameworks[11] can query statuses of tasks for known task IDs. The Mesos master responds with the current state of the nonterminal tasks. On Nomad, the cluster state is captured in the raft stores of the schedulers, and there is no good way to back up the cluster state and restore from a snapshot. Users are expected to resubmit the jobs. Nomad can then reconcile the cluster state, which creates a lot of churn in services.

Conclusion

Designing for failures in all aspects of a distributed cluster scheduler is a must for operational stability and reliability. Scheduler agents should be developed with the understanding that only finite amounts of resources exist on a given system. Processes could leak resources or consume more resources than they were intended to, resulting in unexpected performance degradation caused by resource contention. These scheduler agents must also be able to converge on a good state by using robust reconciliation mechanisms during a given failure (or set of failures), even when particular failure modes could inundate the scheduler with cluster events—for example, the loss of many agent systems caused by a power failure.

Engineers looking to build scheduling systems should consider all failure modes of the underlying infrastructure they use and consider how operators of scheduling systems can configure remediation strategies, while aiding in keeping tenant systems as stable as possible during periods of troubleshooting by the owners of the tenant systems.

The cutting-edge nature of this field of engineering makes it one of the most exciting areas in which to work, enabling workload mobility, uniform scalability, and self-healing systems to become widespread. C

Related articles on queue.acm.org

Hadoop Superlinear Scalability
Neil Gunther, Paul Puglia, and Kristofer Tomasette
https://queue.acm.org/detail.cfm?id=2789974

A Conversation with Phil Smoot
https://queue.acm.org/detail.cfm?id=1113332

The Network is Reliable
Peter Bailis and Kyle Kingsbury
https://queue.acm.org/detail.cfm?id=2655736

References

1. Apache Mesos; http://mesos.apache.org/documentation/latest/frameworks/.
2. Envoy; https://www.envoyproxy.io/.
3. Fluentd; https://www.fluentd.org/.
4. Ionel, G., Schwarzkopf, M., Gleave, A., Watson, R.N.M. and Hand, S. Firmament: Fast, centralized cluster scheduling at scale. In *Proceedings of the 12th Usenix Symposium on Operating Systems Design and Implementation*, 2016; https://research.google.com/pubs/pub45746.html.
5. Isard, M., Prabhakaran, V., Currey, J., Wieder, U., Talwar, K. and Goldberg, A. Quincy: Fair scheduling for distributed computing clusters. *Proceedings of the 22nd ACM SIGOPS Symposium on Operating System Principles*, 2009, 261–276; https://dl.acm.org/citation.cfm?id=1629601.
6. Krasnov, V. On the dangers of Intel's frequency scaling. Cloudflare, 2017; https://blog.cloudflare.com/on-the-dangers-of-intels-frequency-scaling/.
7. Lo, D., Cheng, L., Govindaraju, R., Ranganathan, P. and Kozyrakis, C. Improving resource efficiency at scale with Heracles. *ACM Trans. Computer Systems 34*, 2 (2016); http://dl.acm.org/citation.cfm?id=2882783.
8. Microsoft System Center. Methods and formula used to determine server capacity. TechNet Library, 2013; https://technet.microsoft.com/en-us/library/cc181325.aspx.
9. Rosen, R. Understanding the new control groups API. LWN.net, 2016; https://lwn.net/Articles/679786/.
10. Schwarzkopf, M., Konwinski, A., Abd-El-Malek, M. and Wilkes, J. Omega: Flexible, scalable schedulers for large compute clusters. In *Proceeding of SIGOPS 2013 European Conference on Computer Systems*; https://research.google.com/pubs/pub41684.html.
11. Verma, A., Pedrosa, L., Korupolu, M., Oppenheimer, D., Tune, E. and Wilkes, J. Large-scale cluster management at Google with Borg. *Proceedings of the 10th European Conference on Computer Systems*, 2015; https://dl.acm.org/citation.cfm?id=2741964.

Diptanu Gon Choudhury (@diptanu) works at Facebook on large-scale distributed systems. He is one of the maintainers of the Nomad open source cluster scheduler and previously worked on cluster schedulers on top of Apache Mesos at Netflix.

Timothy Perrett (@timperrett) is an infrastructure engineering veteran, author, and speaker and has led engineering teams at a range of blue-chip companies. He is primarily interested in scheduling systems, programming language theory, security systems, and changing the way industry approaches software engineering.

DOI:10.1145/3190562

Article development led by acmqueue
queue.acm.org

A look at JavaScript libraries in the wild.

BY TOBIAS LAUINGER, ABDELBERI CHAABANE, AND CHRISTO B. WILSON

Thou Shalt Not Depend on Me

MANY WEBSITES USE third-party components such as JavaScript libraries, which bundle useful functionality so that developers can avoid reinventing the wheel. jQuery (https://jquery.com/) is arguably the most popular open source JavaScript library at the moment; it is found on 84% of the most popular websites as determined by Amazon's Alexa (https://www.alexa.com/topsites). But what happens when libraries have security issues? Chances are that websites using such libraries inherit these issues and become vulnerable to attacks.

Given the risk of using a library with known vulnerabilities, it is important to know how often this happens in practice and, more importantly, who is to blame for the inclusion of vulnerable libraries—the developer of the website, or maybe a third-party advertisement, or tracker code loaded on the website?

We set out to answer these questions and found that with 37% of websites using at least one known vulnerable library, and libraries often being included in quite unexpected ways, there clearly is room for improvement in library handling on the Web. To that end, this article makes a few recommendations about what can be done to improve the situation.

JavaScript Vulnerabilities

Before delving into how to detect the use of vulnerable libraries on the Web, we need to agree on what constitutes a vulnerability. First, we are interested only in code that will run on the client side—that is, in a Web browser. JavaScript is the de facto standard language for that purpose, and it has become notorious for security vulnerabilities such

This article is based on original research by T. Lauinger, A. Chaabane, S. Arshad, W. Robertson, C. Wilson, and E. Kirda. It was first published as "Thou Shalt Not Depend on Me: Analysing the Use of Outdated JavaScript Libraries on the Web" in *Proceedings of the 2017 Network and Distributed System Security Symposium*; https://seclab.ccs.neu.edu/static/publications/ndss2017jslibver.pdf).

as XSS (cross-site scripting), which allows an attacker to inject malicious code (or HTML) into a website. In particular, if a JavaScript library accepts input from the user and does a poor job validating it, an XSS vulnerability might creep in, and all websites using this library could become vulnerable.

As an example, consider jQuery's `$()` function. It has different behavior depending on which type of argument is passed: if the argument is a string containing a CSS (Cascading Style Sheets) selector, the function searches the DOM (Document Object Model) tree for corresponding elements and returns references to them; if the input string contains HTML, the function *creates* the corresponding elements and returns the references. As a consequence, developers who pass improperly sanitized input to this function may inadvertently allow attackers to inject code into the page even though the programmer's intent is to select an existing element. While this API design places convenience over security considerations, and the implications could be better highlighted in the documentation, it does not automatically constitute a vulnerability in the library.

In older versions of jQuery, however, the `$()` function's leniency in parsing string parameters could lead to complications by misleading developers to believe, for example, that any string beginning with # would be interpreted as a selector and could be safe to pass to the function, as `#test` selects the element with the identifier `test`. Yet, jQuery considered parameters containing an HTML `<tag>` *anywhere* in the string as HTML (https://bugs.jquery.com/ticket/9521), so that a parameter such as `#<img src=/ onerror=alert(1)>` would lead to code execution rather than a selection. This behavior was considered a vulnerability and fixed.

Other vulnerabilities in JavaScript libraries include cases where libraries fail to sanitize inputs that are expected to be pure text but are passed to `eval()` or `document.write()` internally, which could cause them to be executed as script or rendered as markup. Attackers could exploit these capabilities to steal data from a user's browsing session, initiate transactions on the user's behalf, or place fake content on a website. Therefore, it is important that JavaScript libraries do not introduce any new attack vectors into the websites where they are used.

A drawback of detecting a library by its hash is that it cannot be detected when there is no corresponding reference file in the catalogue.

At the time of our research, there was no single "authoritative" public database of JavaScript vulnerabilities. We manually searched the Open Source Vulnerability Database (OSVDB), the National Vulnerability Database (NVD), public bug trackers, GitHub comments, blog posts, and the list of vulnerabilities detected by Retire.js (https://retirejs.github.io/retire.js/) to gather metadata about vulnerable and fixed versions for the 11 popular libraries shown in Figure 1. As a result, given the name of one of these 11 libraries and a specific release version, we can say whether we know about any publicly disclosed vulnerability—but there are likely more vulnerabilities that we do not know about. Thus, what we report here should be seen as a lower bound.

Library Detection

Collecting vulnerability metadata manually was feasible because we restricted ourselves to 11 of the most popular libraries. For detection of libraries used on websites, however, an automated approach was needed. At first, detecting a library on a website does not sound too complicated: check how the library file is called in the official distribution, such as `jquery-3.2.1.js`, and look for that name in the URLs loaded by websites. Unfortunately, it's rarely that easy. Web developers can rename files, and they do. Using this simple strategy rather than the more complex detection methodology would miss 44% of all URLs containing the Modernizr library, for example. This is not acceptable.

Our approach uses a combination of static and dynamic methods. The static method is a slight improvement over the name-based approach: instead of detecting library files by their name, we detect them by the file hash. This required a comprehensive catalogue of library file hashes, compiled from download links found on the libraries' websites, and on JavaScript CDNs (content delivery networks) maintained by Google, Microsoft, and Yandex, as well as the community-based CDNs jsDelivr, cdnjs, and OSS CDN. Some libraries, such as Bootstrap and jQuery, maintain their own branded CDNs, which were included as well. All versions and variants of each library were downloaded. Variants typically included the "debug" version of the source code with comments, and a "minified" production version that had whitespace removed and internal identifiers shortened for smaller file size and faster page-load times.

A drawback of detecting a library by its hash is that it cannot be detected when there is no corresponding reference file in the catalogue. This can happen, for example, when Web developers modify the source code of the file. Source-code modifications such as addition or removal of comments, or custom minification, occur quite frequently in practice. Out of a random sample of scripts encountered in our crawls that were known to contain jQuery, only 15% could be detected based on the file hash. Therefore, we complemented the static detection with a dynamic detection method.

Dynamic detection examines the runtime environment when the library is loaded in a Web browser. Many libraries register as a window-global variable and make available an attribute that contains the version number of the library. On a website using jQuery, for example, typing `$.fn.jquery` into the developer console of the browser returns a version number such as 3.2.1. Only detections returning a standard three-component `major.minor.patch` version number as used in semantic versioning (http://semver.org/) are counted. By convention, the major version component is increased for breaking changes, the minor component for new functionality, and the patch component for backward-compatible bug fixes. Discarding detections with invalid or empty version attributes reduces the number of false-positive detections—that is, detections that do not actually correspond to the use of a library.

Furthermore, for the purposes of our data analysis, the version number of each detected library instance is needed to look up whether any vulnerabilities are known. Unfortunately, some libraries do not programmatically export version attributes, some libraries added this feature only in more recent versions, and some library loading techniques such as Browserify or Webpack may prevent the library from registering its window-global variable. Furthermore, since only one instance of a window-global variable can exist at any time, when a library is loaded multiple times in the same page, only the last instance is visible at runtime. All these cases result in false-negative detections—that is, the dynamic-detection signature does not detect the library, even though it is present in a website.

Combining the static and dynamic detection methods overcomes their respective limitations. Our research paper also describes an offline variant of dynamic detection, used for the corner case of duplicate library inclusions.

Causality Trees

An important aspect of our research was finding out who is to blame for the inclusion of vulnerable libraries. To that end, we needed to model causal resource inclusion relationships in websites in order to represent how a library was included in a page. For example, a library may be referenced directly in a Web page, or it can be included transitively when another referenced script loads additional resources. We call this model *causality trees*.

A causality tree contains a directed edge `A → B` if and only if element *A* causes element *B* to load. The elements modeled for this study are scripts and embedded HTML documents. A relationship exists whenever an element creates another element or changes an existing element's URL. Examples include a script creating an iframe, and a script changing the URL of an iframe.

While the nodes in a causality tree correspond to nodes in the website's DOM, their structure is entirely unrelated to the hierarchical DOM tree. Rather, nodes in the causality tree are snapshots of elements in the DOM tree at *a specific point in time* and may appear multiple times if the DOM elements are repeatedly modified. For example, if a script creates an iframe with URL U1 and later changes the URL to U2, the corresponding script node in the causality tree will have two document nodes as its children, corresponding to URLs U1 and U2 but referring to the same HTML `<iframe>` element. Similarly, the predecessor of a node in the causality tree is not necessarily a predecessor of the corresponding HTML element in the DOM tree; they may even be located in two different HTML documents, such as when a script appends an element to a document in a different frame.

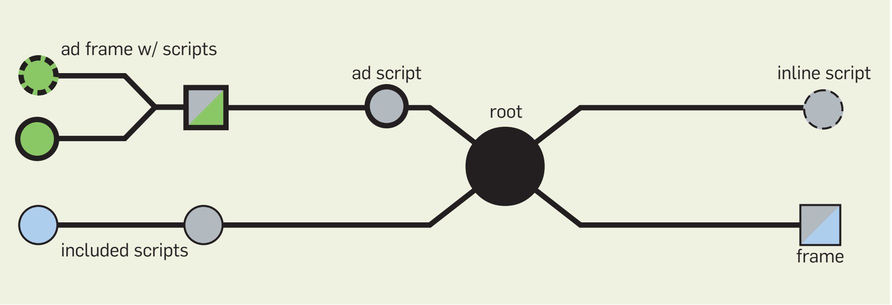
Figure 2 shows a synthetic example of a causality tree. The large black circle is the document root (main document), filled circles are scripts, and squares are HTML documents (for example, embedded in frames). Edges denote "created by" relationships; for example, in Figure 2 the main document includes the gray script, which in turn includes the blue script. Dashed lines around nodes denote inline scripts, while solid lines denote scripts included from an URL. Thick outlines denote that a resource was included from a known ad network, tracker, or social widget.

The color of nodes in Figure 2 denotes which document they are at-

Figure 1. Popular libraries with known vulnerabilities.

Angular
Backbone
Dojo
Ember
Handlebars
jQuery
jQuery-Migrate
jQuery-Mobile
jQuery-UI
Mustache
YUI 3

Figure 2. Generic example of a causality tree.



tached to in the DOM: gray corresponds to resources attached to the main document, while one of four colors is assigned to each further document in frames. Document squares contain the color of their parent location in the DOM, and their own assigned color. Resources created by a script in one frame can be attached to a document in another frame, as shown by the gray script that has a blue child in Figure 2 (that is, the blue script is a child of the blue document in the DOM).

Figure 3a shows a LinkedIn widget as included in the causality tree of `mercantil.com`. (An interactive version is available online at https://seclab.ccs.neu.edu/static/projects/javascript-libraries/.) Note the Web developer embedded code provided by the social network into the main document, which in turn initializes the widget and creates several scripts in multiple frames.

Web Crawl

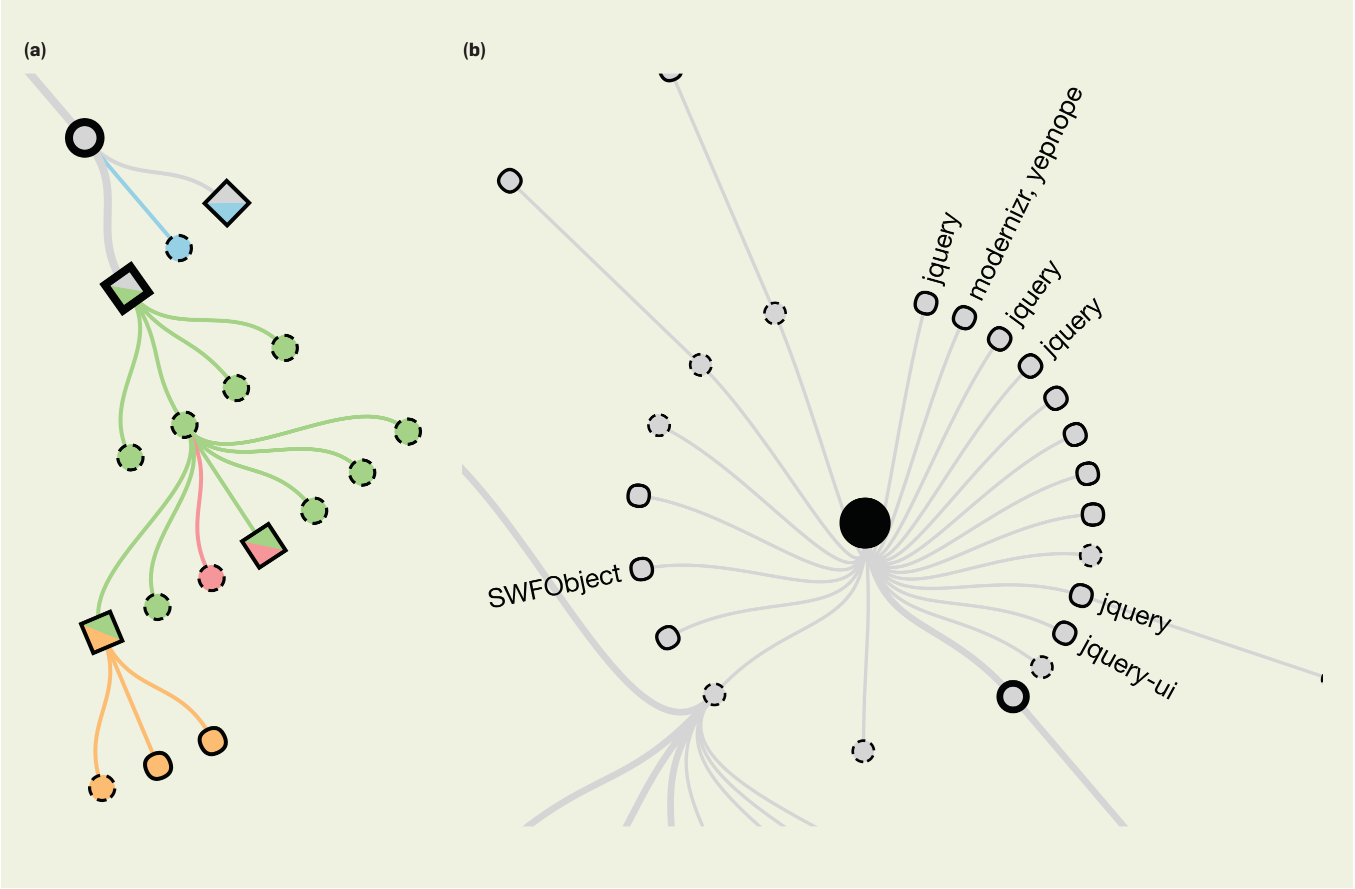
Causality trees are generated using an instrumented version of the Chromium Web browser. Its Chrome DevTools Protocol (https://chromedevtools.github.io/devtools-protocol/) allows detection of most resource-inclusion relationships; for some corner cases, we had to resort to source code modifications in the browser. We also link library detections to nodes in the causality tree and run a modified version of AdBlock Plus to label (but not block) advertisement, tracking, and social media nodes in the causality trees. While visiting a page, the crawler scrolls downward to trigger loading of any dynamic content. As page-loaded events proved to be unreliable, our crawler remains on each page for a fixed delay of 60 seconds before clearing its entire state, restarting, and then proceeding to the next site.

To gain a representative view of JavaScript library usage on the Web, we collected two different datasets. First, we crawled Alexa's top 75,000 domains, which represent popular websites. Second, we crawled 75,000 domains randomly sampled from a snapshot of the .com zone—that is, a random sample of all websites with a .com address, which was expected to be dominated by less popular websites. The two crawls, conducted in May 2016, successfully generated causality trees for the homepages of 71,217 domains in Alexa and 62,086 domains in .COM. Failures resulted from timeouts and unresolvable domains, which were expected especially for .COM since the zone file contains domains that may not have an active website.

How Websites Use Libraries …

Overall, our study used static and dynamic signatures for 72 open source libraries. We found at least one library on the homepage of 87% of the Alexa sites and 65% of the .COM sites. Figure 4 shows the 12 most common libraries in Alexa. jQuery is by far the most popular, used by 84% of the Alexa sites and 61% of the .COM sites. In other words, nearly every website that is using a library is using jQuery. SWFObject, a library used to include Adobe Flash content, is ranked seventh (4%) and 10th (2%), despite being discontinued since 2013. On the other hand, several relatively well-known libraries such as D3, Dojo, and Leaflet appear below the top 30 in both crawls, possibly because

Figure 3. Causality tree of `Mercantile.com`.



they are less commonly used on the homepages of websites.

While the majority of libraries used in Alexa are hosted on the same domain as the website, most inclusions are loaded from external domains in .COM. In the case of jQuery, 59% of all inclusions in Alexa websites are internal, and 39% are external. The remainder are inline inclusions where the source code of the library is not loaded from a file but directly wrapped in `<script> // library code here </script>` tags. Only 30% of the websites in the .COM crawl host jQuery internally, whereas 68% rely on external hosting. This highlights a difference in how larger and smaller websites include libraries.

In both crawls, JavaScript CDNs are among the most popular domains from which libraries are loaded. In Alexa, almost 18% of library files are loaded from ajax.googleapis.com, Google's JavaScript CDN (13% in .COM), followed by jQuery's branded CDN code.jquery.com (4% in Alexa, 3% in .COM). The less popular sites in the .COM crawl, however, also frequently load libraries from domains related to domain parking and hosting providers.

When looking at *why* libraries are included, it turns out that around 3% of jQuery inclusions in Alexa and almost 26% in .COM are caused by advertisement, tracking, or social media widget code. For SWFObject, more than 42% of inclusions in Alexa come from ads. In other words, the blame for including a now-unsupported library does not go directly to those websites but to the ad networks they are using. Advertisement, tracking, or social media widget code is typically provided by an external service and loaded *as is* by the website developer—who may not be aware that the included code will load additional libraries and who has no say in which versions of these libraries will be loaded. Overall, libraries loaded by ads can be found on 7% of sites in Alexa, and on 16% of sites in .COM.

An important aspect of our research was finding out who is to blame for the inclusion of vulnerable libraries.

... And How They Include *Vulnerabilities*

We compiled metadata about vulnerable versions of the 11 libraries shown in Figure 1. Among the Alexa sites, 38% use at least one of these 11 libraries in a version known to be vulnerable, and 10% use two or more different known vulnerable versions. In .COM, the vulnerability rates are slightly lower—37% of sites have at least one known vulnerable library, and 4% two or more—but the sites in .COM also have a lower rate of library use in general. As a result, those .COM sites that *do* use a library have a higher probability of vulnerability than those in Alexa.

Looking at individual libraries shows that known vulnerable versions can make up a majority of all uses of those libraries in the wild. jQuery, for example, has around 37% known vulnerable inclusions in Alexa, and 55% in .COM. Angular has 39%–40% vulnerable inclusions in both crawls, and Handlebars has 87%–88%. This does not mean, however, that Handlebars is "more vulnerable" than jQuery; it means only that Web developers use known vulnerable versions more often in the case of Handlebars than for jQuery. The emphasis here is on *known vulnerable*, as each library may contain vulnerabilities that are *not* known. In that sense, these results are a lower bound on the use of vulnerable libraries.

So far, we have examined whether sites are potentially vulnerable—that is, whether they include one or more known vulnerable libraries—and how that adds up on a per-library level. Now let's return to our analysis of *how* libraries are included by sites. Figure 5 shows two prominent factors that are connected to a higher fraction of vulnerable inclusions:

- Inline inclusions of jQuery have a clearly higher fraction of vulnerable versions than internally or externally hosted copies.
- Library inclusions by ad, widget, or tracker code appear to be more vulnerable than unrelated inclusions. While the difference is relatively small for jQuery in Alexa, the vulnerability rate of jQuery associated with ad, widget, or tracker code in .COM—89%—is almost double the rate of unrelated inclusions. This may be a result of less reputable ad networks or widgets being used on the smaller sites in .COM as opposed to the larger sites in Alexa.

At this point, a word about the limitations of our study is in order. We do not check whether a known vulnerability in a library can be exploited when used on a specific website. If Web developers can ensure a library vulnerability cannot be exploited on their site, they do not need to update to a newer version. Yet, as we will discuss, the release notes of libraries rarely contain enough information to allow a non-expert to decide whether continuing to use a vulnerable library on a specific site is safe or not. Therefore, in practice, the safe course of action would be always to update when a vulnerability in a library is discovered.

Unfortunately, because of the release cycles and patching behavior of library maintainers, updating a library dependency is easier said than done. Only a very small fraction of sites using vulnerable libraries (less than 3% in Alexa, and 2% in .COM) could become free of vulnerabilities by applying only patch-level updates. Updates of the least significant version component, such as from `1.2.3` to `1.2.4`, would generally be expected to be backward compatible. In most cases, however, patch updates are not available. The vast majority of sites would need to install at least one library with a more recent major or minor version to remove all vulnerabilities. Migrating to these newer versions might necessitate additional code changes and site testing because of incompatibilities in the API.

Beyond vulnerabilities and considering all 72 supported libraries, 61% of Alexa sites and 46% of .COM sites are at least one patch version behind on one of their included libraries. Even though such updates should be "painless," they are often neglected. Similarly, the median Alexa site uses a version released 1,177 days (1,476 days for .COM) before the newest available release of the library. These results demonstrate that the majority of Web developers are working with library versions released a long time ago. Time differences measured in years suggest that Web developers rarely update their library dependencies once they have deployed a site.

Analyzing the use of JavaScript libraries on websites reveals that libraries are often used in unexpected ways. For example, about 21% of the websites including jQuery in Alexa, and 17% in .COM, do so two or more times in a single Web page. That alone is no cause for concern; when a website contains `<iframe>`s with documents loaded from different origins, it may even be necessary to include the library multiple times because of the same-origin policy limiting scripts' access across origins. Yet, a closer look reveals that 4% of websites using jQuery in Alexa include the same version of the library two or more times in the *same* document (5% in .COM), and 11% (6%) include two or more *different* versions of jQuery in the same document. No benefit is derived by including the library multiple times in the same document because jQuery registers itself as a window-global variable. Unless special steps are taken, only the last loaded and executed instance in each document can be used by client code; the other instances will be hidden. Asynchronously included instances may even create a race condition, making it difficult to predict which version will prevail in the end.

The development practices adopted by library maintainers have a big influence on how difficult it will be for library users to keep their dependencies up to date.

As an illustration, consider the detail from the causality tree for `mercantil.com` in Figure 3b. The site includes jQuery four times. All these inclusions are referenced directly in the main page's source code, some of them directly adjacent to each other. On other sites, duplicate inclusions were caused by multiple scripts transitively including their own copies of jQuery. While we can only speculate on why these cases occur, at least some of them may be related to server-side templating, or the combination of independently developed components into a single document. Indeed, we have observed cases where a Web application (for example, a WordPress plug-in) that bundled its own version of a library was integrated into a page that already contained a separate copy of the same library. Since duplicate inclusions of a library do not necessarily break any functionality, many Web developers may not be aware that they are including a library multiple times, and even fewer may be aware that the duplicate inclusion may be potentially vulnerable.

What Can, and *Should*, Be Done?

Our research has shown that vulnerable libraries are widely used on the Web. A number of factors are at play, and no single actor can be made responsible for the situation. Instead, let's look at it from three different angles.

Dependency management. Website

developers need to be aware of which libraries they are using. It is too easy to forget about a library when it is manually copied into the codebase. Instead, we recommend explicitly declaring a project's dependencies in a central location. For client-side JavaScript, Bower (https://bower.io/) was one of the first dependency management tools. Yarn (https://yarnpkg.com/) is a more recent entry to the scene, backed by the repository of NPM (Node Package Manager; https://www.npmjs.com/), which contains not only server-side Node.js packages, but also client-side JavaScript libraries. Explicit dependencies make it easy to automatically include the library code of the declared version into the project. Additionally, tools such as Retire.js (https://retirejs.github.io/retire.js/), AuditJS (https://github.com/OSSIndex/auditjs), or Snyk (https://snyk.io/) can scan the declared dependencies for known vulnerable versions. Ideally, Web developers should make such tools part of their build process, so that attempts to include a known vulnerable library cause a build to fail. For projects where such a proactive approach is not an option, Retire.js also has a browser extension that can detect vulnerable libraries in deployed websites.

Library development. The development practices adopted by library maintainers have a big influence on how difficult it will be for library users to keep their dependencies up to date. To that end, we conducted an informal survey of the 12 most frequently used libraries (Figure 4).

Before developers can update the libraries they are using, they must be made aware that there is a need to update. None of these 12 libraries, however, seems to maintain a mailing list or other dedicated channel for security announcements. Some libraries have Twitter accounts, but these contain a lot of additional "noise" unrelated to new releases or security issues. None of the libraries appears to systematically allocate CVE (Common Vulnerabilities and Exposures) numbers or register security issues in popular vulnerability databases. Only Angular prominently highlights patched vulnerabilities in the release notes of new library versions; the other libraries often mention unspecific "security fixes" along with a long list of other changes, if they are mentioned at all.

In addition to the difficulty of finding out about vulnerabilities, it is very rare to find information about the range of versions affected by a vulnerability. Given this general lack of readily available information, security-conscious users of a library do not have much of a choice other than to update every time a new version is released. Updating is often "painful," however, for a number of reasons ranging from the short release cycles common in Web library development to breaking API changes and the need for testing after each library update.

To end this survey on a positive note, we highlight the security practices followed by Ember (https://emberjs.com). Its maintainers commit to patching long-term support releases so that library users do not need to deal with frequent breaking API changes. Ember maintains a security announcement mailing list, registers CVE numbers, mentions security issues in release notes, lists the range of versions affected by a vulnerability, and provides a dedicated email address to report security issues. These practices ease the burden of dealing with vulnerabilities. Let's hope that other library maintainers will follow suit.

Third-party components. The previous paragraphs assumed that website developers directly include libraries, which makes it their responsibility to keep them up to date. The results of the Web crawls, however, show that this assumption often does not hold in practice. In fact, many website developers load external scripts such as advertisements, tracker code, or social media widgets. These third-party components sometimes include libraries on their own. This study has shown that such behavior may cause duplicate inclusions of a library, and that these indirect inclusions come with a higher rate of vulnerability. Under some circumstances, sandboxing the third-party code in an iframe may be an option to limit the damage. In general, however, website developers must rely on the maintainers of these components to update their code.

Conclusion

Most websites use JavaScript libraries, and many of them are known to be vulnerable. Understanding the scope of the problem, and the many unexpected ways that libraries are included, are only the first steps toward improving the situation. The goal here is that the information included in this article will help inform better tooling, development practices, and educational efforts for the community.

Figure 4. Top 12 libraries by frequency in Alexa.

	ALEXA	COM
jQuery	84%	61%
jQuery-UI	24%	8%
Modernizr	21%	9%
Bootstrap	13%	5%
jQuery-Migrate	11%	11%
Underscore	6%	2%
SWFObject	4%	2%
Moment	4%	1%
RequireJS	3%	2%
jQuery-Form	3%	3%
Backbone	3%	2%
Angular	2%	2%

Figure 5. Vulnerable fraction of JQuery inclusions.

	ALEXA	COM
All Inclusions	37%	55%
Internal	38%	42%
External	35%	63%
Inline	55%	90%
Ad/Widget/Tracker	38%	89%
No Ad/Widget/Tracker	37%	46%

Related articles on queue.acm.org

Dismantling the Barriers to Entry
Rich Harris
https://queue.acm.org/detail.cfm?id=2790378

JavaScript and the Netflix User Interface
Alex Liu
https://queue.acm.org/detail.cfm?id=2677720

MongoDB's JavaScript Fuzzer
Robert Guo
https://queue.acm.org/detail.cfm?id=3059007

Tobias Lauinger is a Ph.D. student at Northeastern University with an interest in Internet-scale measurements of everything security and beyond.

Abdelberi Chaabane is a security researcher at Nokia Bell Labs whose work focuses on empirical large-scale studies to measure and understand online threats.

Christo B. Wilson is an associate professor at Northeastern University whose work focuses on security and privacy on the Web, and algorithmic transparency.

practice

DOI:10.1145/3190572

Article development led by acmqueue
queue.acm.org

A.B.A. = Always be automating.

BY THOMAS A. LIMONCELLI

Documentation Is Automation

LET ME TELL you about two systems administrators I know. Both were overloaded, busy IT engineers. Both had many repetitive tasks to do. Both wanted to automate these tasks. After observing these two people for a year, I noticed that one made a lot of progress, while the other one didn't. It was not a matter of skill—both were very good software engineers. The difference was their approach, or mind-set.

I would say the successful one had a mindset of always thinking in terms of moving toward the goal of a better automated system. Imagine an analog gauge that points to the left when measuring that a process is completely manual but slides to the right as progress is made toward a fully autonomous system. The developer mindset is always intent on moving the needle to the right.

The less successful person didn't write much code, and he had excellent reasons why: I'm too busy! The person who made the request can't wait! I have 100 other things to do today! Nobody's allocating time for me to write code!

The successful person had the same pressures but somehow managed to write a lot of code. The first time he did something manually, he documented the steps. That may not be code in the traditional sense, but writing the steps in a bullet list is similar to writing pseudocode before writing actual code. It doesn't run on a literal computer, but you run the code in your head. You are the CPU.

Automation is putting process into code. A bullet list in a process document is code if it is treated that way.

The second time the successful engineer did something manually, he followed his own documentation. This might seem strange since he knew the process well enough to document it, but by following his own documentation, he found opportunities to improve it. He made correc-

IMAGE BY PABLO PRAT

tions and augmented the command-line snippets he had recorded.

As he repeated this process over and over, the document evolved to be much better. The example names and numbers in the command lines were replaced by variables. Ambiguous statements such as "make sure everything is OK" were replaced by checklists of things to be tested, which were soon augmented by commands that performed the tests.

Soon this manual process was feeling more and more like real automation. There was less thinking, more following orders. Doing the process "manually" was more like copying and pasting command-line snippets from the document and pasting them in his terminal window. I call this PasteOps.

By doing this in the open, with collaborative document systems such as a wiki or Git repository, coworkers are able to join in: Mary fixes a command line that broke on a certain class of machine. Joe does some Web searches and soon a step that previously required a mouse click is replaced by a command.

As more and more coworkers adopt this work style, the entire team contributes to the constant goal of better automation.

This engineer, who began with the same time pressure and other obstacles as the other less successful engineer, has not yet written any traditional code, so to speak, but the process has become much more automated.

In the future these smatterings of command-line snippets will be combined into one big program that automates the entire process. This tool will be used as the basis for a Web-based self-service portal. This will allow users to do the task on demand, seven days a week, even when the sysadmins are asleep.

Meanwhile, the other engineer, the one who was "too busy to write code," is no closer to getting started.

The difference between these two engineers is that one is willing to do work manually just to get the task done. The other is willing to do work manually only as a mechanism for generating artifacts (documentation and code snippets) that "move the needle" toward an automated world.

A Culture of Automating

People who are successful at automating tasks tend to work this way in every aspect of their jobs. It is just how they work; it is part of their culture.

The successful engineer has a quick way to create documents for new procedures and to find existing procedures. People with this mindset avoid the cognitive load of deciding whether or not a task is worth documenting, because they document everything. On finding a potential improvement, they are not slowed by the speed bump of switching from a document viewer to a document editor because they work from the editor at the start. Heck, they have a dedicated second monitor just for their editing app!

People with this culture revise documents in real time. Meanwhile, the less successful engineer has a stack of notes that he honestly plans on entering into a document someday soon—perhaps the same "someday" when he will start writing code.

The successful engineer realizes that the earlier he starts collaborating, the sooner others can contribute. Together they can create a culture of documentation that spreads throughout the team. Thus, every project is collaborative and has a "stone soup" feeling, as all are invited to bring their skills and insights. The more people who embody this culture, the more success it has.

This culture can be summarized in two sentences: (1) Every manual action must have a dual purpose of completing a task and improving the system. (2) Manual work should not be tolerated unless it generates an artifact or improves an existing one.

Four Phases

Traditional software development involves requirements gathering and so on. In the culture of automation, we wiggle and iterate among four overlapping phases: document the steps; create automation equivalents; create automation; and create self-service and autonomous systems.

Phase 1: Document the Steps

At the start, developers perform a task manually to learn the process. They keep good notes and record what they do for each step. This is often exploratory or may require interviewing experts on how to do the process. They produce an artifact—written documentation describing how the process is done.

Beginning programmers are taught to write a program in pseudocode first, then turn each line of pseudocode into actual code. The same applies to automation: if you can't describe the process in writing, you can't automate it.

Documentation is automation. Following a step-by-step guide is automation: you are the CPU; you are following the instructions. As with any prototyping language, you should not expect perfection, but learning. You have the benefit of being able to spot and fix problems along the way. You are a CPU that improves the code as it executes!

There is no reason to wait for the document to be perfect before moving on to the next phase. All that is required is that the people involved gain the minimum necessary confidence in the document to move forward.

Phase 2: Create Automation Equivalents

As the document matures, manual action generates a new kind of artifact: command-line snippets. The document is augmented with automated equivalents for each step.

At first, you simply paste the command line used to perform the step into the document as is. The next time you manually perform the task, you improve it—perhaps by rewriting it. Over time these command examples become fully functional code snippets.

Other improvements happen. Mouse clicks are replaced by commands. Quality assurance steps are added then automated.

Mouse clicks and other GUI actions that have no API or command-line equivalent are noted. Bugs are filed with the vendor and the bug ID is added to the document. (As a manager, I am unsatisfied when engineers tell me, "Oh, the vendor knows that's a problem," but can't show me a bug ID. I say, "Bug ID, or it didn't happen.")

Yes, the process is still being done manually, but now each manual iteration is done by setting variables and pasting lines of commands into the terminal window. Each manual iteration tests the accuracy of the snippets and finds new edge cases, bugs, and better ways of verifying the results.

Phase 3: Create Automation

Soon these command-line snippets are turned into longer scripts. Like all good code, this is kept in a source code repository. The artifacts begin looking more like real software.

Perhaps the code is performing only certain steps or works in only a narrow set of circumstances. Each manual iteration, however, expands the code to cover new use cases. No manual iteration should leave the scripts unimproved. In fact, it should be the other way around. Each manual iteration is simply a test for the most recent improvements. You should look forward to finding an edge case that breaks the code because this is an opportunity to fix the problem.

Often the entire process is more complex than is appropriate for a scripting language. Turning snippets of PowerShell or Bash into standalone scripts is easy, but it is often better to write larger programs in languages such as Python, Ruby, or Go. The individual snippets usually translate easily, and when they don't, a reasonable stopgap measure is to have the program "shell out" to run the command line. These can be "downcoded" into the native language later as needed.

Since February 2015, the SRE (site reliability engineering) team at Stack Overflow has switched from a mixture of Python and Bash to Go. Even though Go isn't a scripting language, for small programs it compiles and runs nearly as fast as Python takes to start. At Stack Overflow we tend to prefer compiled, type-checked languages for large programs, especially when multiple people are collaborating, and, therefore, no one person is familiar with every line of code. Our policy was that Bash scripts couldn't be larger than 100 lines and Python programs couldn't be larger than 1,000 lines. Those seemed like reasonable limits. Rewriting scripts when they grew beyond the limit, however, was a lot of work. It was better to start in Go and avoid the conversion.

Phase 4: Self-Service and Autonomous Systems

In the next phase the script becomes a stand-alone tool, which then becomes part of a larger system, usually with a Web-based front end. Ideally, some kind of self-service portal can be created so that users can activate the automation themselves. Even better is to create an autonomous system. The difference between *automated* and *autonomous* is the difference between a *tool* that someone can use to create new user accounts, and a *system* that monitors the HR database and creates and deletes accounts without human intervention. Autonomous systems eliminate the human task.

Depending on how frequently the task is needed, this phase may not be worth the effort. The return on investment may indicate that stopping at the tool stage is sufficient. CI (continuous integration) systems such as Jenkins and runbook automation systems such as Rundeck, however, make it easy to create simple, RBAC (role-based access control) restrained, self-service portals.

Discipline

Maintaining this culture and not backsliding takes discipline. Every manual iteration must move you closer to better automation.

It is tempting to revert to the old methods or skip updating the documentation "just this once" because you are in a hurry, or you'll fix it next time, or the new system is broken, or you're not in a good mood today. The developer mindset, however, resists such temptations and treats every manual iteration as an opportunity that should not be squandered.

Doing something manually "because it is faster" is often a sign that engineers feel pressure, but they do not realize they are mortgaging their future. In reality, the old way may feel faster only because they are more comfortable with it. Often the time pressure they feel does not actually exist. Will the person who asked the engineer to do this particular task notice that it took 20 minutes instead of five minutes? If the person is in the middle of a two-hour meeting, he or she certainly won't notice. A few extra minutes spent improving the system, however, pays off in all future iterations.

Manual work should not be tolerated unless it generates an artifact or improves an existing one.

On the contrary, I have often debugged brittle code in front of the requester. I give the requester a choice: Fix their problem quickly the manual way, or allow me to take a little extra time and fix the automation so that all can benefit. We agree in advance to a deadline at which point I'll revert to the reliable-but-manual process. Technology and non-technical people alike jump at the opportunity. I suspect they do this because it feels good to help fix a larger problem, but also because it has entertainment value.

When it is tempting to revert to the old way for expediency's sake, it is useful to remind yourself that the benefit of automation is not always speed. Automation that is slower but less error-prone can be a net gain if the errors take a long time to fix. Preventing a single error that requires a day of restoring data from backups could be invaluable. Because I'm fat-fingered and easily distracted, this is a major motivation for me.

Another benefit is the consistency that automation can bring. Increased variation increases the cost of support and makes other automation projects more burdensome by increasing the number of edge cases. For example, at one site I discovered that half the Linux systems used raw disk partitions, while the others used Linux LVM (Logical Volume Manager) to manage disk storage. This complicated the monitoring system (which now had to handle both variations), procedure documentation (which had to be written and tested with both variations), and so on. Tasks that should have taken minutes took hours (or days) on the machines that could not benefit from LVM's flexibility. The two variations did not exist for technical reasons. The installation process was not automated, and the manual process resulted in what I will politely call "creativity," where we would have preferred conformity.

Automation and documentation democratize the work, lowering the bar so that others may do the task. Any positive progress through the four phases enables more people on a team to do a task, thus enabling you to distribute work among your peers and reduce single points of failure. You might be the only person with the knowledge and experience to do the task, but a little documentation can empower others to do it instead, even if they don't have a deep understanding of the technology. Even if the documentation covers only the most common situation and is full of warnings such as "This procedure won't work if the user has [insert technical details]" or "If you get the following error, don't try to fix it yourself. Call Mary or Bob." Future updates to the document can cover those edge cases. You don't need everyone on the team to have your years of experience, just the wisdom to follow directions and contact you if they get stuck.

These benefits save you time in ways other than just making the process faster. They make you more efficient, reduce the work for the entire team by reducing the complexity that must be managed, or create a workforce multiplier that enables other people to take work off your plate.

By creating a culture of continuous improvement, constantly taking baby steps along the four phases, the work becomes less stressful and easier to manage. While the other reasons listed here are quite logical, what motivates me to maintain this discipline is more emotional: I want to reduce stress and have more time for creativity and joy.

By creating a culture of continuous improvement, constantly taking baby steps along the four phases, the work becomes less stressful and easier to manage.

The Leftover Principle

Focusing on automating the easy parts means the work left for humans is the difficult stuff. That means automation just made life worse for you.[2] Ironic, eh? Weren't computers supposed to make life easier? This is called the Leftover Principle, as discussed in this column in 2015.[3]

The solution to this is the Compensatory Principle: People and machines should each do what they are good at and not attempt what they don't do well. That is, each group should compensate for the other's deficiencies.[1]

Therefore, rather than focusing on automating what's easy, focus on automating the boring parts (unlike you, computers love repetition), the difficult parts (reduce error-prone steps), and the parts that need to happen when you would rather be asleep. As a human, you are better than computers at improvisation and being flexible, exercising judgment, and coping with variations. So, don't fret over not being able to automate deciding which of four paths to take when that decision is purely a judgment call. Instead, automate the four paths but leave the selection process to you!

Documentation as automation lowers the bar for what can be automated, enabling you to improve tasks you would have avoided in both the Leftover Principle and the Compensatory Principle.

Ambiguous Requirements

The computer scientists reading this piece might be wondering why I'm not recommending a formal requirements-gathering stage or other more rigorous software-engineering best practices.

The reality is that an organization's IT environment is usually so opaque and amorphous that requirements cannot be written beyond a basic statement of desired results. The first time one attempts to use an API call is more a matter of trial and error than following instructions. Nothing works the first time. It is hours (or days) of guesswork, exploration, and discovery. Nearly every operating system, framework, and IT system contribute to this mess. IT does not live in a world of high school physics where one has the luxury of an infinitely large, flat, frictionless surface. IT lives in a world that is a squishy swamp of vendor promises and "damned if you do, damned if you don't" choices, all made worse by authentication systems that seem to be designed to work only on sunny days.

Early in the discovery process it is not obvious exactly what to do, what will work, or how long it will take to code. It is more exploration than rote execution. It reminds me of a framed sign that hung in my father's chemistry lab that read, "If we knew what we were doing, it wouldn't be called research."

As a result, an incremental and iterative approach is required. Early phases are more exploratory, and later phases are more confident. You start by working on the low-hanging fruit, not because they are easy, but because if you are honest with yourself, you have to admit to having no idea how the more difficult parts could ever conceivably be implemented. By doing the easier parts, however, you gain the experience that makes the other parts possible. Initial experiences inform later decisions, build confidence, and give you the fortitude to continue. Soon the impossible parts of the project become possible.

Therefore, working in a waterfall approach is untenable. Maintaining a lockstep workflow through the phases would mean never leaving the first gate. Some steps may be ready for full automation, while others lag behind. You cannot wait for the documentation to be perfect before moving to the next phase. You may not have figured out a command-line equivalent for step 46, but the other steps can move forward. I once used a system that was pretty darn automated, except someone had to be there to click "ok" at one point. It took months to eliminate that. I'm glad we didn't wait.

Enable Early Collaboration

An iterative structure improves your ability to work collaboratively. If the documentation is on a wiki or similar system, everyone can contribute and update the documentation. Once the basic infrastructure is in place, everyone can fill in the missing pieces by adding support for new edge cases, improving testing, and so on. Good engineers build the initial framework but make it easy for others to contribute. I call this the "stone soup" method of software development: you bring the cooking pot and everyone else fills it.

The earlier you share, the better. The earlier you can enable this collaboration, the sooner more people can contribute. For example, by keeping the documentation in something easy to edit, such as a wiki or Git repository, everyone on the team can "be the CPU," not only testing the algorithm, but also contributing improvements. The sooner the software is packaged in a way that everyone can use, the sooner feedback is available. Someone with a developer mindset treats the documentation and code a lot like an open source project—available and easy to contribute to.

To enable collaboration, use the same tools people are already using. If your team uses Git, keep the documentation in Git. Repurpose the team's wiki, Google docs structure, CI system, or whatever will lower the bar to contributions.

The anti-pattern is to work privately and plan on releasing the documentation and code to the rest of the team "next week." Next week never comes. It is a red flag when I hear someone say "the code isn't ready to share with other people" or "I can't show the document to the team until the next round of edits." The opposite is true. If you release something you think "works only for you," it enables others to figure out how to make it run for them. How can you know what parts work only for you if you haven't let other people try it?

It is important for managers to create a structure where projects are easily sharable from the start, and to provide (gentle) pressure to move projects into that structure when they are not. I try to role-model the release-early attitude by starting my documentation and code in an open Git repository, unabashedly inserting comments such as "This code sucks and needs to be replaced," or by indicating which parts are missing or could use improvement. Do not shame people for releasing broken code; reward them for transparency and promoting collaboration.

Conclusion

Some IT engineers never have time to automate their work. Others have the same time constraints but succeed in creating the preconditions (documentation, code snippets) that enable automation.

As you work, you have a choice. Will each manual task create artifacts that allow you to accelerate future work, or do you squander these opportunities and accept the status quo?

By constantly documenting and creating code-snippet artifacts, you accelerate future work. That one-shot task that could never happen again, does happen again, and next time it moves faster. Even tasks that aren't worth automating can be improved by documenting them, as documentation *is* automation.

Every IT team should have a culture of constant improvement—or movement along the path toward the goal of automating whatever the team feels confident in automating, in ways that are easy to change as conditions change. As the needle moves to the right, the team learns from each other's experiences, and the system becomes easier to create and safer to operate.

A good team has a structure in place that makes the process frictionless and collaborative—plus, management that rewards and encourages the developer's mindset. Always be automating.

Acknowledgments
This article benefited from feedback from John Allspaw (Adaptive Capacity Labs), Nicole Forsgren (DORA: DevOps Research and Assessment LLC), and Jason Shantz (Stack Overflow, Inc.).

References
1. Allspaw, J. A mature role for automation, part II. Kitchen Soap; https://www.kitchensoap.com/2013/08/20/a-mature-role-for-automation-part-ii/.
2. Bainbridge, L. Ironies of automation. *Automatica 19*, 6 (1983), 775–779;
3. https://pdfs.semanticscholar.org/0713/bb9d9b138e4e0a15406006de9b0cddf68e28.pdf.
4. Limoncelli, T.A. Automation should be like Iron Man, not Ultron. *acmqueue 13*, 8 (2015); https://queue.acm.org/detail.cfm?id=2841313.

Thomas A. Limoncelli is the site reliability engineering manager at Stack Overflow Inc. in New York City. His books include *The Practice of System and Network Administration*, *The Practice of Cloud System Administration*, and *Time Management for System Administrators*. He blogs at EverythingSysadmin.com and tweets @YesThatTom.

DOI:10.1145/3209581

Bias in Web data and use taints the algorithms behind Web-based applications, delivering equally biased results.

BY RICARDO BAEZA-YATES

Bias on the Web

OUR INHERENT HUMAN tendency of favoring one thing or opinion over another is reflected in every aspect of our lives, creating both latent and overt biases toward everything we see, hear, and do. Any remedy for bias must start with awareness that bias exists; for example, most mature societies raise awareness of social bias through affirmative-action programs, and, while awareness alone does not completely alleviate the problem, it helps guide us toward a solution. Bias on the Web reflects both societal and internal biases within ourselves, emerging in subtler ways. This article aims to increase awareness of the potential effects imposed on us all through bias present in Web use and content. We must thus consider and account for it in the design of Web systems that truly address people's needs.

Bias has been intrinsically embedded in culture and history since the beginning of time. However, due to the rise of digital data, it can now spread faster than ever and reach many more people. This has caused bias in big data to become a trending and controversial topic in recent years. Minorities, especially, have felt the harmful effects of data bias when pursuing life goals, with outcomes governed primarily by algorithms, from mortgage loans to advertising personalization.[24] While the obstacles they face remain an important roadblock, bias affects us all, though much of the time we are unaware it exists or how it might (negatively) influence our judgment and behavior.

The Web is today's most prominent communication channel, as well as a place where our biases converge. As social media are increasingly central to daily life, they expose us to influencers we might not have encountered previously. This makes understanding and recognizing bias on the Web more essential than ever. My main goal here is thus to raise the awareness level for all Web biases. Bias awareness would help us design better Web-based systems, as well as software systems in general.

Measuring Bias

The first challenge in addressing bias is how to define and measure it. From a statistical point of view, bias is a systemic deviation caused by an inaccurate estimation or sampling process. As a result, the distribution of a variable could be biased with respect to the original, possibly unknown, distribution. In addition, cultural biases can be found in our inclinations to our shared personal beliefs, while cognitive biases affect our behavior and the ways we make decisions.

Figure 1 shows how bias influences

» key insights

- Any remedy for bias starts with awareness of its existence.
- Bias on the Web reflects biases within ourselves, manifested in subtler ways.
- We must consider and account for bias in the design of Web-based systems that truly address the needs of users.

YOU'RE RIGHT
AND
EVERYONE
ELSE IS
WRONG.

Figure 1. The vicious cycle of bias on the Web.

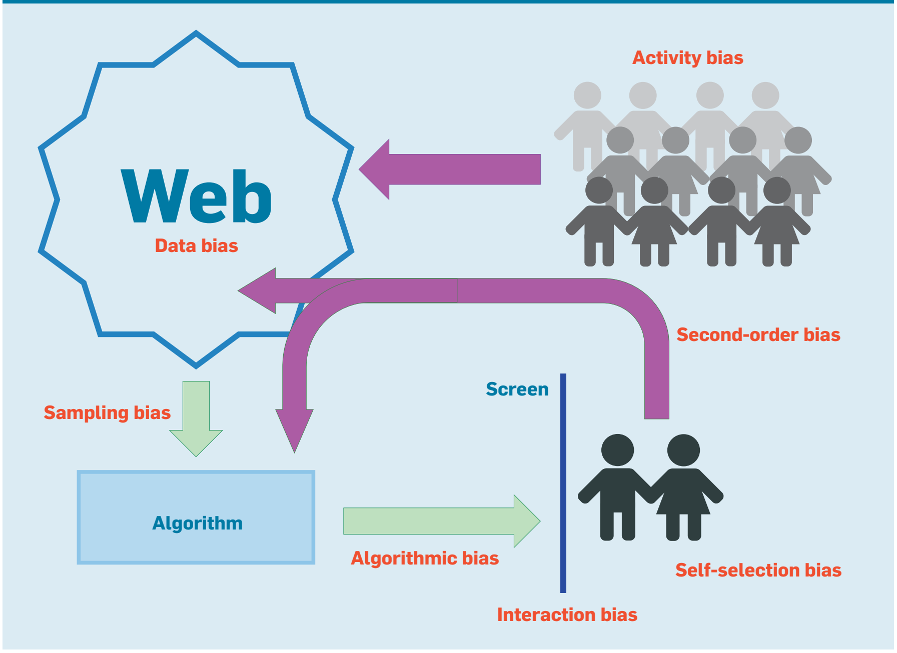
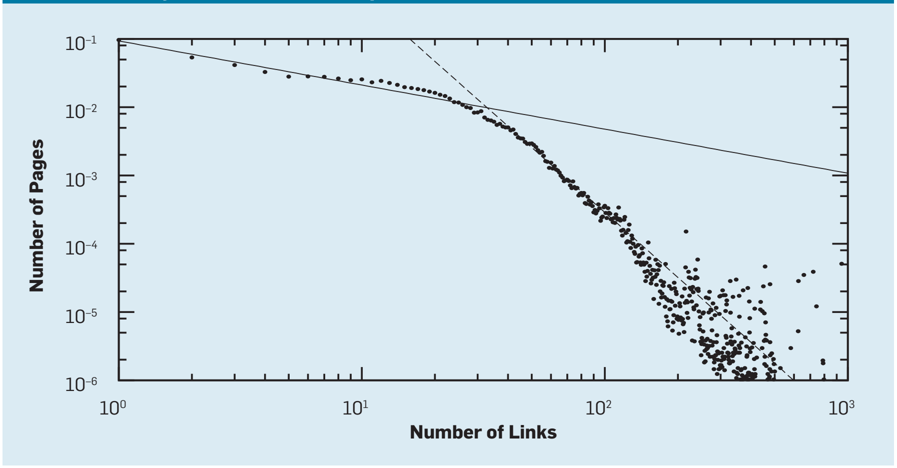


Figure 2. Shame effect (line with small trend direction) vs. minimal effort (notable trend direction) on number of links on U.K. webpages, with intersection between 12 and 13 links. Data at far right is probably due to pages having been written by software, not by Web users or developers.[5]



both the growth of the Web and its use. Here, I explain each of the biases (in red) and classify them by type, beginning with activity bias resulting from how people use the Web and the hidden bias of people without Internet access. I then address bias in Web data and how it potentially taints the algorithms that use it, followed by biases created through our interaction with websites and how content and use recycles back to the Web or to Web-based systems, creating various types of second-order bias.

Consider the following survey of research on bias on the Web, some I was involved with personally, focusing on the significance of the categories of bias identified, not on methodological aspects of the research. For more detail, see the References and the research listed in the online appendix "Further Reading" (dl.acm.org/citation.cfm?doid=3209581&picked=formats) of this article.

Activity Bias, or Wisdom of a Few

In 2011, a study by Wu et al.[28] on how people followed other people on Twitter found that the 0.05% of the most popular people attracted almost 50% of all participants;[28] that is, half of the Twitter users in the dataset were following only a few select celebrities. I thus asked myself: What percentage of active Web users generate half the content in a social media website? I did not, however, consider the silent majority of Web users who only *watch* the Web without contributing to it, which in itself is a form of self-selection bias.[14] Saez-Trumper and I[8] analyzed four datasets, and as I detail, the results surprised us.

Exploring a Facebook dataset from 2009 with almost 40,000 active users, we found 7% of them produced 50% of the posts. In a larger dataset of Amazon reviews from 2013, we found just 4% of the active users. In a very large dataset from 2011 with 12 million active Twitter users, the result was only 2%. Finally, we learned that the first version of half the entries of English Wikipedia was researched and posted by 0.04% of its registered editors, or approximately 2,000 people, indicating only a small percentage of all users contribute to the Web and the notion that it represents the wisdom of the overall crowd is an illusion.

In light of such findings,[8] it did not make sense that just 4% of the people voluntarily write half of all the reviews in the Amazon dataset. I sensed something else is at play. A month after publication of our results, my hunch was confirmed. In October 2015, Amazon began a corporate campaign against paid fake reviews that continued in 2016 by suing almost 1,000 people accused of writing them. Our analysis[8] also found that if we consider only the reviews that some people find helpful, the percentage decreases to 2.5%, using the positive correlation between the average helpfulness of each review according to users and a proxy of text quality. Although the example of English Wikipedia is the most biased, it represents a *positive* bias. The 2,000 people at the start of English Wikipedia probably triggered a snowball effect that helped Wikipedia become the vast encyclopedic resource it is today.

Zipf's least-effort principle,[29] also called Zipf's law, maintains that many people do only a little while few people do a lot, possibly helping explain a big part of activity bias. However, economic and social incentives also play a role in yielding this result. For example, Zipf's law can be seen in most Web measures (such as number of pages per website or number of links per webpage). Figure 2 plots the number of links in U.K. webpages on the *x*-axis and the number of webpages on the *y*-axis. Zipf's law is clearly visible on the right side, in the line with the more negative slope. However, there is a strong social force at the beginning of the *x*-axis I call the "shame effect" that makes the slope less negative. It also illustrates that many people prefer to exert the least effort, though most people also need to feel they do enough to avoid feeling ashamed of their effort.[5] These two effects are common characteristics of people's activity on the Web.

Finally, Nobel laureate Herbert Simon said, "A wealth of information creates a poverty of attention." Activity bias thus generates a "digital desert" across the Web, or Web content no one ever sees. A lower bound comes from Twitter data where Saez-Trumper and I[8] found that 1.1% of the tweets were written and posted by people without followers. Reviewing Wikipedia use statistics gave us an upper bound, whereby 31% of the articles added or modified in May 2014 were never visited in June. The actual size of the digital desert on the Web likely lies in the first half of the 1% to 31% range.

On the other hand, bias is not always negative. Due to activity bias, all levels of Web caching are highly effective at keeping the most used content readily available, and the load on websites and the Internet network in general is then much lower than would be potentially possible.

Data Bias

As with people skills, data quality is heterogeneous and thus, to some extent, expected to be biased. People working in government, universities, and other institutions that deal with information should publish data of higher quality and less bias, while social media as a whole is much larger, biased, and without doubt, of lower average quality. On the other hand, the number of people contributing to social media is probably at least one order of magnitude greater than the number of people working in information-based institutions. There is thus more data of *any* quality coming from all people, including high-quality data, no matter what definition of what quality one uses. Still, a lot of fake content on the Web seems to spread faster than reliable content.[17]

The first set of biases seen in people interacting with the Web is due to their demographics. Accessing and using the Internet correlates with educational, economic, and technological bias, as well as other characteristics, causing a ripple effect of bias in Web content and links. For example, it is estimated that over 50% of the most popular websites are in English, while the percentage of native English speakers in the world is approximately only 5%; this increases to 13% if all English speakers are included, as estimated by Wikipedia. Geographical bias is also seen in Web content associated with large cities and tourist attractions. Another example of the network effect of Web bias is the link structure of the Web itself. Figure 3 plots the number of links from the Web within Spain to other countries, along with exports from Spain to the same other countries.[3] The countries toward the bottom right are outliers, as they had all sold the right to use their domains for other purposes (such as the .fm country code, top-level domain for the Federated States of Micronesia). Ignoring them, the correlation between exports and number of links is more than 0.8 for Spain. In fact, the more developed a country is, the greater is the correlation, ranging from 0.6 for Brazil to 0.9 for the U.K.[4]

Figure 3. Economic bias in links for the Web in Spain.[3]

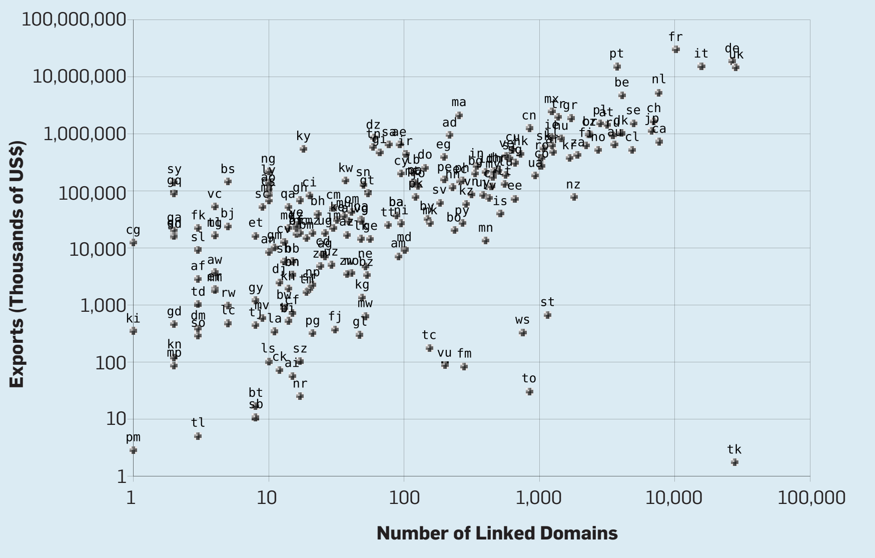
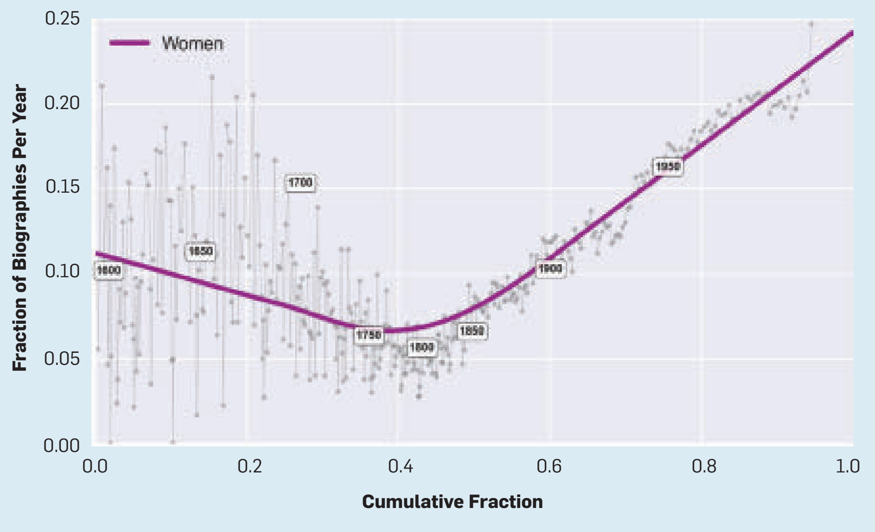


Figure 4. Accumulated fraction of women's biographies in Wikipedia.[16]



A second set of biases is due to the interaction between different types of bias. Consider Figure 4, which plots the fraction of biographies of women in Wikipedia,[16] a curve that could be explained through systemic gender bias throughout human history.[25] However, an underlying factor hides a deeper bias that is revealed when looking more closely at the creation process. In the category of biographies, Wikipedia statistics show that less than 12% of Wikipedia editors are women. In other categories, gender bias is even worse, reaching 4% in geography. On the other hand, as the percentage of all publicly reported Wikipedia female editors is just 11%, biographies actually show a small positive bias. Keep in mind these values are also biased, as not all Wikipedia editors identify their gender, and females might thus be underrepresented.

Our third source of data bias is Web spam, a well-known human-generated malicious bias that is difficult to characterize. The same applies to content (near) duplication (such as mirrored websites) that, in 2003, represented approximately 20% of static Web content.[13]

Since measuring almost any bias is difficult, its effect on prediction algorithms using machine learning are likewise difficult to understand. As Web data represents a biased sample of the population to begin with, studies based on social media may have a significant amount of error we can be sure is not uniformly distributed. For the same reason, the results of such research cannot be extrapolated to the rest of the population; consider, for example, the polling errors in the 2016 U.S. presidential election,[18] though online polls predicted the outcome better than live polls. Other sources of error include biased data samples (such as due to selection bias) or samples too small for the analytical technique at hand.[7]

Algorithmic Bias and Fairness

Algorithmic bias is added by the algorithm itself and not present in the input data. If the input data is indeed biased, the output of the algorithm might also reflect the same bias. However, even if all possible biases are detected, defining how an algorithm should proceed is generally difficult, in the same way people disagree over what is a fair solution to any controversial issue. It may even require calling on a human expert to help detect if an output indeed includes any bias at all. In a 2016 research effort that used a corpus of U.S. news to learn she-he analogies through word embeddings, most of the results was reported as biased, as in nurse-surgeon and diva-superstar instead of queen-king.[9] A quick Web search showed that approxi-

Figure 5. Heat maps of eye-tracking analysis on web-search results pages, from 2005 (left) to 2014 (right).[18]

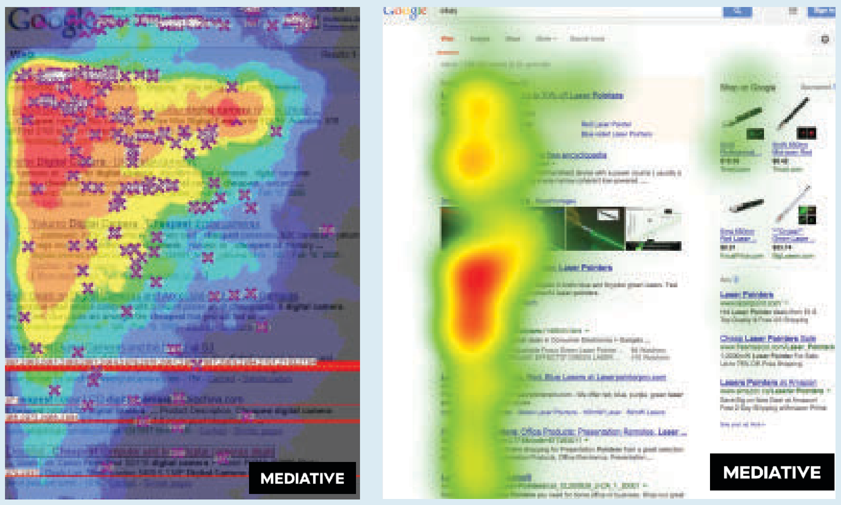
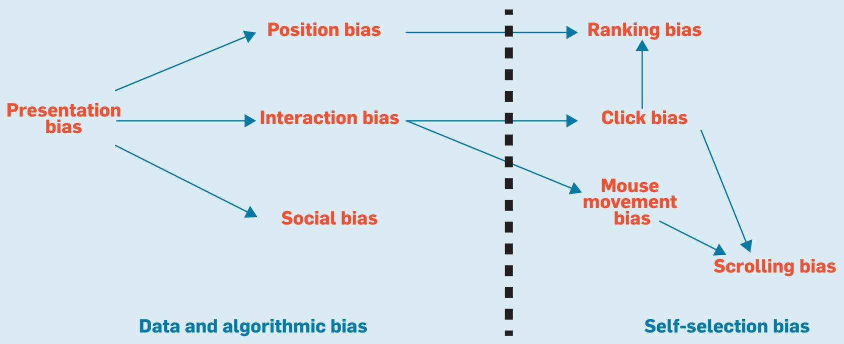


Figure 6. Dependency graph of biases affecting user interaction.



Possible classification of biases whereby the cultural and cognitive columns are user-dependent.

Bias Type	Statistical	Cultural	Cognitive
Algorithmic	●	?	?
Presentation	●		
Position	●		
Sampling	●		
Data	●	●	●
Second-order	●	●	●
Activity		●	●
User Interaction		●	●
Ranking		●	●
Social		●	●
Self-selection			●

mately 70% of influential journalists in the U.S. were men, even though at U.S. journalism schools, the gender proportions are reversed. Algorithms learning from news articles are thus learning from texts with demonstrable and systemic gender bias. Yet other research has identified the presence of other cultural and cognitive biases.[10,22]

On the other hand, some Web developers have been able to limit bias. "De-biasing" the gender-bias issue can be addressed by factoring in the gender subspace automatically.[9] Regarding geographical bias in news recommendations, large cities and centers of political power surely generate more news. If standard recommendation algorithms are used, the general public likely reads news from a capital city, not from the place where they live. Considering diversity and user location, Web designers can create websites that give a less centralized view that also shows local news.[15]

"Tag recommendations," or recommending labels or tags for items, is an extreme example of algorithmic bias. Imagine a user interface where a user uploads a photo and adds various tags, and a tag recommendation algorithm then suggests tags that people have used in other photos based on collaborative filtering. The user chooses the ones that seem correct, enlarging the set of tags. This sounds simple, but a photo-hosting website should not include such functionality. The reason is that the algorithm needs data from people to improve, but as people use recommended tags, they add fewer tags of their own, picking from among known tags while not adding new ones. In essence, the algorithm is doing prolonged hara-kiri on itself. If we have a "folksonomy," or tags that come only from people, websites should not themselves recommend tags. On the other hand, many websites use this idea to provide the ability to search similar images through related tags.

Another critical class of algorithmic bias in recommender systems is related to what items the system chooses to show or not show on a particular webpage. Such bias affects user interaction, as explored next. There is ample research literature on all sorts of algorithmic bias; see the online appendix for more.

In addition to the bias introduced by interaction designers, users have their own self-selection bias.

Bias on User Interaction

One significant source of bias is user interaction, not only on the Web, but from two notable sources: the user interface and the user's own self-selected, biased interaction. The first is "presentation bias," whereby everything seen by the user can get clicks while everything else gets no clicks. This is particularly relevant in recommendation systems. Consider a video-streaming service in which users have hundreds of recommendations they can browse, though the number is abysmally small compared to the millions that could potentially be offered. This bias directly affects new items or items that have never been seen by users, as there is no usage data for them. The most common solution is called "explore and exploit," as in Agarwal et al.,[2] who studied a classical example applied to the Web. It exposes part of user traffic to new items randomly intermingled with top recommendations to explore and, if chosen, exploit usage data to reveal their true relative value. The paradox of such a solution is that exploration could imply a loss or an opportunity cost for exploiting information already known. In some cases, there is even a loss of revenue (such as from digital ads). However, the only way to learn and discover (new) good items is exploration.

"Position bias" is the second bias. Consider that in western cultures we read from top to bottom and left to right. The bias is thus to look first toward the top left corner of the screen, prompting that region to attract more eyes and clicks. "Ranking bias" is an important instance of such bias. Consider a Web search engine where results pages are listed in relevant order from top to bottom. The top-ranked result will thus attract more clicks than the others because it is both the most relevant and also ranked in the first position. To avoid ranking bias, Web developers need to de-bias click distribution so they can use click data to improve and evaluate ranking algorithms.[11,12] Otherwise, the popular pages become even more popular.

Other biases in user interaction include those related to user-interaction design; for example, any webpage where a user needs to scroll to see additional content will reflect bias like presentation bias. Moreover, content near images has a greater probability of being clicked, because images attract user attention. Figure 5 shows examples from eye-tracking studies whereby, after universal search (multiple types of answers) is introduced, the non-text content counteracts position bias in the results page;[18] it also shows the advertising column on the right would attract additional attention.

Social bias defines how content coming from other people affects our judgment. Consider an example involving collaborative ratings: Assume we want to rate an item with a low score and see that most people have already given it a high score. We may increase our score just thinking that perhaps we are being too harsh. Such bias has been explored in the context of Amazon reviews data[26] and is often referred to as "social conformity," or "the herding effect."[20]

Finally, the way a user interacts with any type of device is idiosyncratic. Some users are eager to click, while others move the mouse to where they look. Mouse movement is a partial proxy for gaze attention and thus a computationally inexpensive replacement for eye tracking. Some of us may not notice the scrolling bar, others prefer to read in detail, and yet others prefer just skim. In addition to the bias introduced by interaction designers, users have their own self-selection bias. White[27] explored a good example of how cultural and cognitive biases affect Web search engines, showing that users tend to choose answers aligned with their existing beliefs.

To make bias even more complex, interaction biases cascade through the system, and Web developers have great difficulty trying to isolate them. Figure 6 outlines an example of how such biases cascade and depend on one another, implying that Web developers are always seeing their combined effects. Likewise, users who prefer to scroll affect how they move the mouse, as well as which elements of the screen they are able to click.

Interaction biases are crucial to analyzing the user experience, as well as to a website's overall performance, as many Web systems are optimized through implicit user feedback. As such optimized systems are increasingly based in machine learning, they learn to reinforce their own biases or the biases of other linked systems, yielding sub-optimal solutions and/or self-fulfilling prophecies. These systems sometimes even compete among themselves, such that an improvement in one results from degradation of another that uses a different (inversely correlated) optimization function. A classic example is the tension between improving the user experience and increasing monetization (such as the way increasing numbers of ads generally diminishes the user experience).

As any attempt to be unbiased might already be biased through our own cultural and cognitive biases, the first step is thus to be aware of bias.

Vicious Cycle of Bias

Bias begets bias. Imagine we are a blogger planning our next blog post. We first search for pages about the topic we wish to cover. We then select a few sources that seem relevant to us. We select several quotes from these sources. We write new content, putting the quotes in the right places, citing the sources. And, finally, we publish the new entry on the Web.

This content-creation process does not apply solely to bloggers but also to content used in reviews, comments, social network posts, and more. The problem of drifting off message occurs when a subset of content is selected based on what the search engine being used believes is relevant. The ranking algorithm of the search engine thus biases a portion of a given topic's organic growth on the Web. A study my colleagues and I conducted in 2008[6] found that approximately 35% of the content on the Web in Chile was duplicated, and we could trace the genealogy of the partial (semantic) duplication of those pages. Today, the semantic-duplication effect might be even more widespread and misleading.

The process creates a vicious cycle of second-order bias, as some content providers get better rankings, leading to more clicks; that is, the rich get richer. Moreover, the duplication of content only compounds the problem of distinguishing good pages from bad pages. In turn, Web spammers make use of content from good pages to appear themselves to be quality content, only adding to the problem. So, paradoxically, search engines harm themselves unless they *do not* account for all biases.

Another example of second-order bias comes from personalization algorithms (such as the filter-bubble effect),[21] which do not affect Web content but rather the content exposed to the user. If a personalization algorithm uses only our interaction data, we see only what we want to see, thus biasing the content to our own selection biases, keeping us in a closed world, closed off to new items we might actually like. This issue must be counteracted through collaborative filtering or task contextualization, as well as through diversity, novelty, serendipity, and even, if requested, giving us the other side. This has a positive effect on online privacy because, by incorporating such techniques, less personal information is required.

Conclusion

The problem of bias is much more complex than I have outlined here, where I have covered only part of the problem. Indeed, the foundation involves all of our personal biases. On the contrary, many of the biases described here manifest beyond the Web ecosystem (such as in mobile devices and the Internet of Things). The table here aims to classify all the main biases against the three types of bias I mentioned earlier. We can group them in three clusters: The top one involves just algorithms; the bottom one—activity, user interaction, and self-selection—involves those that come just from people; and the middle one—data and second-order—includes those involving both. The question marks in the first line indicate that each program probably encodes the cultural and cognitive biases of their creators. One antecedent to support this claim is an interesting data-analysis experiment where 29 teams in a worldwide crowdsourcing challenge performed a statistical analysis for a problem involving racial discrimination.[3]

In early 2017, US-ACM published the seven properties algorithms must fulfill to achieve transparency and accountability:[1] awareness, access and redress, accountability, explanation, data provenance, auditability, and validation and testing. This article is most closely aligned with awareness. In addition, the IEEE Computer Society also in 2017 began a project to define standards in this area, and at least two new conferences on the topic were held in February 2018. My colleagues and I are also working on a website with resources on "fairness measures" related to algorithms (http://fairness-measures.org/), and there are surely other such initiatives. All of them should help us define the ethics of algorithms, particularly with respect to machine learning.

As any attempt to be unbiased might already be biased through our own cultural and cognitive biases, the first step is thus to be aware of bias. Only if Web designers and developers know its existence can they address, and if possible, correct them. Otherwise, our future could be a fictitious world based on biased perceptions from which not even diversity, novelty, or serendipity would be able to rescue us.

Acknowledgments

I thank Jeanna Matthews, Leila Zia, and the anonymous reviewers for their helpful comments, as well as for Amanda Hirsch for her earlier English revision. C

References

1. ACM U.S. Public Policy Council. *Statement on Algorithmic Transparency and Accountability.* ACM, Washington, D.C., Jan. 2017; https://www.acm.org/binaries/content/ assets/public-policy/2017_usacm_statement_algorithms.pdf
2. Agarwal, D., Chen, B-C., and Elango, P. Explore/exploit schemes for Web content optimization. In *Proceedings of the Ninth IEEE International Conference on Data Mining* (Miami, FL, Dec. 6–9). IEEE Computer Society Press, 2009.
3. Baeza-Yates, R., Castillo, C., and López, V. Characteristics of the Web of Spain. *Cybermetrics 9*, 1 (2005), 1–41.
4. Baeza-Yates, R. and Castillo, C. Relationship between Web links and trade (poster). In *Proceedings of the 15th International Conference on the World Wide Web* (Edinburgh, U.K., May 23–26). ACM Press, New York, 2006, 927–928.
5. Baeza-Yates, R., Castillo, C., and Efthimiadis, E.N. Characterization of national Web domains. *ACM Transactions on Internet Technology 7*, 2 (May 2007), article 9.
6. Baeza-Yates, R., Pereira, Á., and Ziviani, N. Genealogical trees on the Web: A search engine user perspective. In *Proceedings of the 17th International Conference on the World Wide Web* (Beijing, China, Apr. 21–25). ACM Press, New York, 2008, 367–376.
7. Baeza-Yates, R. Incremental sampling of query logs. In *Proceedings of the 38th ACM SIGIR Conference* (Santiago, Chile, Aug. 9–13). ACM Press, New York, 2015, 1093–1096.
8. Baeza-Yates, R. and Saez-Trumper, D. Wisdom of the crowd or wisdom of a few? An analysis of users' content generation. In *Proceedings of the 26th ACM Conference on Hypertext and Social Media* (Guzelyurt, TRNC, Cyprus, Sept. 1–4). ACM Press, New York, 2015, 69–74.
9. Bolukbasi, R., Chang, K.W., Zou, J., Saligrama, V., and Kalai, A. Man is to computer programmer as woman is to homemaker? De-biasing word embeddings. In *Proceedings of the 30th Conference on Neural Information Processing Systems* (Barcelona, Spain, Dec. 5–10). Curran Associates, Inc., Red Hook, NY, 2016, 4349–4357.
10. Caliskan, A., Bryson, J.J., and Narayanan, A. Semantics derived automatically from language corpora contain human-like biases. *Science 356*, 6334 (Apr. 2017), 183–186.
11. Chapelle, O. and Zhang, Y. A dynamic Bayesian network click model for Web search ranking. In *Proceedings of the 18th International Conference on the World Wide Web* (Madrid, Spain, Apr. 20–24). ACM Press, New York, 2009, 1–10.
12. Dupret, G.E. and Piwowarski, B. A user-browsing model to predict search engine click data from past observations. In *Proceedings of the 31st ACM SIGIR Conference* (Singapore, July 20–24). ACM Press, New York, 2008, 331–338.
13. Fetterly, D., Manasse, M., and Najork, M. On the evolution of clusters of near-duplicate webpages. *Journal of Web Engineering 2*, 4 (Oct. 2003), 228–246.
14. Gong, W., Lim, E.-P., and Zhu, F. Characterizing silent users in social media communities. In *Proceedings of the Ninth International AAAI Conference on Web and Social Media* (Oxford, U.K., May 26–29). AAAI, Fremont, CA, 2015, 140–149.
15. Graells-Garrido, E. and Lalmas, M. Balancing diversity to countermeasure geographical centralization in microblogging platforms. In *Proceedings of the 25th ACM Conference on Hypertext and Social Media* (Santiago, Chile, Sept. 1–4). ACM Press, New York, 2014, 231–236.
16. Graells-Garrido, E., Lalmas, M., and Menczer, F. First women, second sex: Gender bias in Wikipedia. In *Proceedings of the 26th ACM Conference on Hypertext and Social Media* (Guzelyurt, TRNC, Cyprus, Sept. 1–4). ACM Press, New York, 2015, 165–174.
17. Lazer, D.M.J. et al. The science of fake news. *Science 359*, 6380 (Mar. 2018), 1094–1096.
18. Mediative. *The Evolution of Google's Search Results Pages & Effects on User Behaviour.* White paper, 2014; http://www.mediative.com/SERP
19. Mercer, A., Deane, C., and McGeeney, K. *Why 2016 Election Polls Missed Their Mark.* Pew Research Center, Washington, D.C., Nov 2016; http://www.pewresearch.org/fact-tank/2016/11/09/why-2016-election-polls-missed-their-mark/
20. Olteanu, A., Castillo, C., Diaz, F., and Kiciman, E. *Social Data: Biases, Methodological Pitfalls, and Ethical Boundaries.* SSRN, Rochester, NY, Dec. 20, 2016; https://ssrn.com/abstract=2886526
21. Pariser, E. *The Filter Bubble: How the New Personalized Web Is Changing What We Read and How We Think.* Penguin, London, U.K., 2011.
22. Saez-Trumper, D., Castillo, C., and Lalmas, M. Social media news communities: Gatekeeping, coverage, and statement bias. In *Proceedings of the ACM International Conference on Information and Knowledge Management* (San Francisco, CA, Oct. 27–Nov. 1). ACM Press, New York, 2013, 1679–1684.
23. Silberzahn, R. and Uhlmann, E.L. Crowdsourced research: Many hands make tight work. *Nature 526*, 7572 (Oct. 2015), 189–191; https://psyarxiv.com/qkwst/
24. Smith, M., Patil, D.J., and Muñoz, C. *Big Data: A Report on Algorithmic Systems, Opportunity, and Civil Rights.* Executive Office of the President, Washington, D.C., 2016; https://obamawhitehouse.archives.gov/sites/default/files/microsites/ostp/2016_0504_data_discrimination.pdf
25. Wagner, C., Garcia, D., Jadidi, M., and Strohmaier, M. It's a man's Wikipedia? Assessing gender inequality in an online encyclopedia. In *Proceedings of the Ninth International AAAI Conference on Web and Social Media* (Oxford, U.K., May 26–29). AAAI, Fremont, CA, 2015, 454–463.
26. Wang, T. and Wang, D. Why Amazon's ratings might mislead you: The story of herding effects. *Big Data 2*, 4 (Dec. 2014), 196–204.
27. White, R. Beliefs and biases in Web search. In *Proceedings of the 36th ACM SIGIR Conference* (Dublin, Ireland, July 28–Aug. 1). ACM Press, New York, 2013, 3–12.
28. Wu, S., Hofman, J.M., Mason, W.A., and Watts, D.J. Who says what to whom on Twitter. In *Proceedings of the 20th International Conference on the World Wide Web* (Hyderabad, India, Mar. 28–Apr. 1). ACM Press, New York, 2011, 705–714.
29. Zipf, G.K. *Human Behavior and the Principle of Least Effort.* Addison-Wesley Press, Cambridge, MA, 1949.

Ricardo Baeza-Yates (rbaeza@acm.org) is Chief Technology Officer of NTENT, a search technology company based in Carlsbad, CA, USA, and Director of Computer Science Programs at Northeastern University, Silicon Valley campus, San Jose, CA, USA.

Watch the author discuss his work in this exclusive *Communications* video. https://cacm.acm.org/videos/bias-and-the-web

DOI:10.1145/3148051

By focusing on users' abilities rather than disabilities, designers can create interactive systems better matched to those abilities.

BY JACOB O. WOBBROCK, KRZYSZTOF Z. GAJOS, SHAUN K. KANE, AND GREGG C. VANDERHEIDEN

Ability-Based Design

RECALL THE LAST time you took a trip out of town. Perhaps you were traveling to a conference far from home. Remember the many forms of transportation you endured: cars, buses, airplanes, and trains. Not only were you responsible for moving yourself over a great distance, you had to move your things as well, including books and baggage. Remember the cramped spaces, sharp elbows, body aches, and exhaustion. Feel again your desire to simply be at your destination with your possessions intact . . .

Such journeys remind us of our physical embodiment in the physical world, that much of our lived experience is fundamentally physical, and that we must contend with the world on physical terms. As computing professionals, we might be tempted to forget this, as our keystrokes summon data instantly from across the globe. But as humans, we still interact with that data through physical devices and displays using our physical senses and bodies. We and the world interact physically.

Civilization's story of technological progress is in no small part the story of an increasingly built physical environment, from the pyramids to roads to skyscrapers to sanitation systems. Much of our energy, collectively and individually, goes into moving and shaping material for such purposes, altering the physical landscape and our movement through it. Some of our most thrilling experiences come by way of changing our bodies' relation to that landscape: bungee jumping, skydiving, scuba diving, and riding a rollercoaster all provide radically new experiences for our bodies in the world.

As designers and builders of interactive systems for human use, we also play a central role in defining people's relationship to and experience of the physical world.[2,13,30] When we design things, we take mere ideas, things without form, and embody them in the world, whether simple sketches or cardboard mockups. They could be pixels on a screen or functioning digital devices. Regardless of the medium, to design and build things is to embody ideas that are then encountered and

» key insights

- **Ability-based design is a new design approach for interactive systems that focuses on people's abilities in context, on what people *can* do, rather than on what they *cannot* do.**
- **Ability-based design scrutinizes the "ability assumptions" behind the design of interactive systems, shifting the responsibility of enabling access from users to the system.**
- **People's abilities may be affected not just by disabilities but by disabling situations; designing for abilities in context leads to more usable, accessible systems for all people.**

used by other embodied people.

This design-and-build activity is profound. It was not long ago in human history that giving form to the formless was considered the purview of the divine. In fact, the English verb "to create" comes from the Latin "creare," which means to bring "form out of nothing." When we design and build systems, we bring form out of nothing.

Unfortunately, unlike the divine, we cannot anticipate all the ways our designs will affect the people who encounter them. And when a mismatch arises, the world can become a very rigidly embodied place (see Figure 1).

Many of the great breakthroughs in interactive computing have come as improved embodiments capable of transforming the way people experience the digital world. Sutherland's interactive display and light pen in SketchPad,[31] Engelbart's and English's mouse in NLS,[4] and Apple's iPhone all represent breakthrough embodiments. But a vital engineering insight is that they, as with all interactive technologies, include certain "ability assumptions" that must be met by human users. These assumptions are often unstated but alienating if they cannot be met.

An everyday example makes the point. In the student union building at the University of Washington in Seattle, wall-mounted touchscreens function as information kiosks for visitors (see Figure 2). In the on-screen operating instructions, a particular word stands out—"just," as in, "*just* touch the screen." In fact, touching the screen requires many abilities, including closing one's hand, extending one's index finger, elevating one's arm, seeing the target, landing accurately, holding steady, and lifting without sliding—along with the ability to read and understand the instructions in the first place. There is clearly no "just" about it.

Where do ability assumptions come from? Designers and developers make assumptions from their own abilities, from the ones they imagine other people have, or the ones of the supposed "average user."[22] Unfortunately, each source of such assumptions is flawed. The first two are prone to bias and unrepresentative; the third, insidious for its statistical façade, does not reflect the diversity of human life.

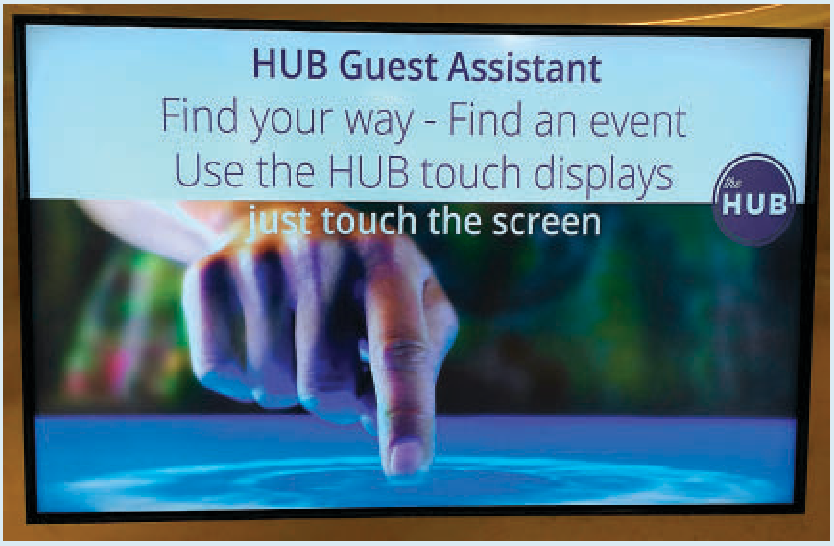
On that point, Rose[25] offered an anecdote from the U.S. Air Force. After World War II, it frequently lost pilots and planes in peacetime crashes—incredibly, 17 on one particular day—so it decided to redesign its cockpits to reduce "pilot error." Air Force engineers measured 4,063 pilots along 140 dimensions, averaging these values to create cockpits to fit the mathematically average pilot. But a young Air Force scientist, Lt. Gilbert Daniels, questioned this approach. He took just 10 of the most important dimensions, added a tolerance of 30% of their ranges around their means, and compared every individual pilot to them to see how many of the 4,063 pilots aligned. The surprising result? Zero. Even among pilots recruited for their congruity, human diversity dictated that individual differences ruled. Only when the Air Force created pilot-configurable cockpits covering the 5th to 95th percentile of pilot measurements did the crashes decline.

Motivated by a need to make interactive computing systems that better match users' abilities, we formulated "ability-based design,"[37,38] aiming to create accessible technologies for people with disabilities and for people in disabling situations (such as in the dark or while walking in the cold or encumbered). Following our work on adaptive user interfaces[9–11] and technologies for people on the go,[15,24,32,33] ability-based design pursues an ambitious vision—that anyone, anywhere, at any time can interact with systems that are ideally suited to their situated abilities, and that the systems do the work to achieve this fit. Here, we expound this vision and describe the steps we have taken toward achieving it.

Figure 1. A person in a wheelchair facing a flight of concrete stairs.

Does the challenge lie in his inability to walk or in the stairs' requirement that he must? Regardless, people should expect and receive more accommodation from their interactive computing systems than they do from an immutable flight of concrete stairs.

Figure 2. A wall-mounted touchscreen instructing users to "*just* touch the screen," though a great many abilities are required to do so.



Ability and Disability

It helps to be explicit about the term "ability." For our purpose, a useful definition comes from the Oxford dictionary: "Possession of the means or skill to *do* something"[a] (emphasis ours). The focus is on acting in the world, not just thinking about it.

Defining the term "disability" is thornier. In 1976, the World Health Organization (WHO) defined disability as, "Any restriction or lack ... of ability to perform an activity in the manner or within the range *considered normal for a human being*"[39] (emphasis ours). Thankfully, in 2001, this normative language yielded to the International Classification of Functioning, Disability, and Health,[b] authored and adopted by WHO, identifying disability as a complex interaction among an individual, activity, society, and the environment, both social and physical. Indeed, research has illuminated just how much social factors play a role in the experience of disability.[28,29]

When considering disability, ability-based design goes further. If "ability" is about having the means or skill to do something, then "disability" means simply being unable to do something. Disability becomes something one experiences rather than something someone has or is. Following such a view, everyone experiences disability, because everyone lacks the means or skill to do quite a few things, at least in certain circumstances. Designing for abilities applies to all people.

We call this perspective the "positive affirmation of ability," namely that all people have abilities, some more than others, and designers and

a https://en.oxforddictionaries.com/definition/ability

b http://www.who.int/classifications/icf/en/

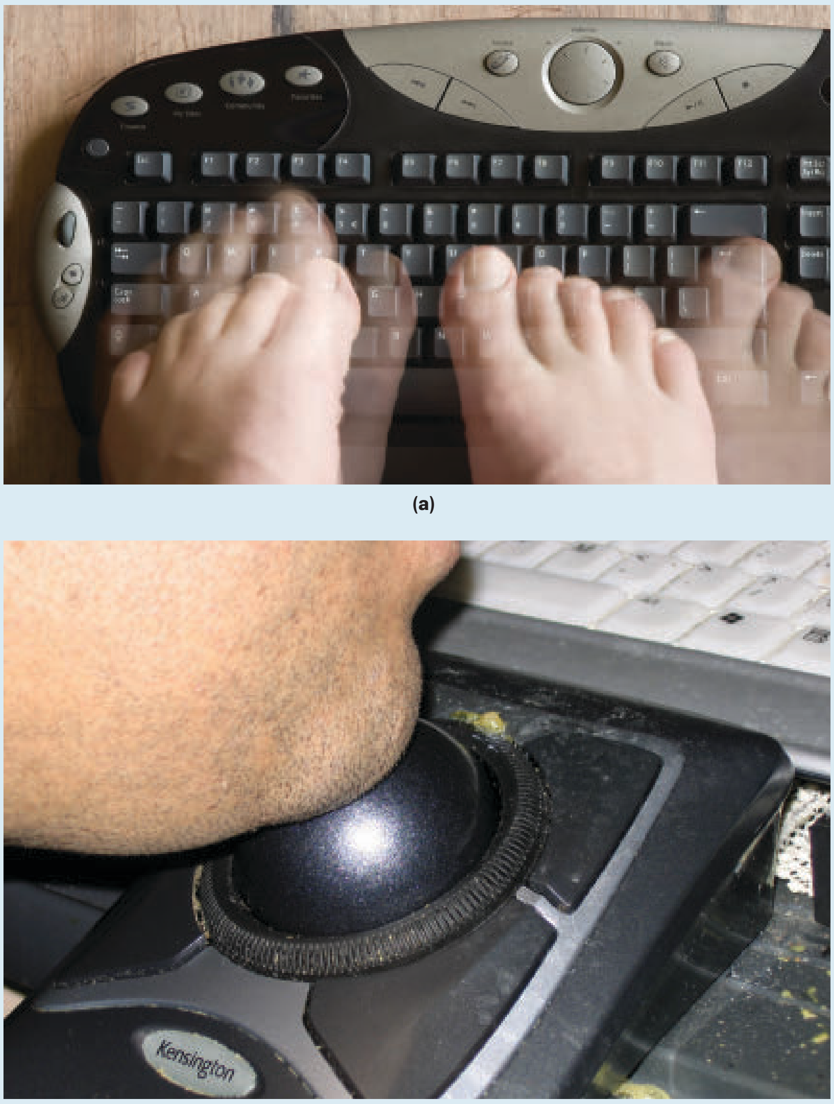
developers ought to create systems for people with abilities of all kinds and degrees. Likewise, Newell[22] referred to "extra-ordinary abilities," saying, "common sense and observation show us that every human being has . . . abilities, some of which can be described as 'ordinary' and some of which are very obviously extra-ordinary." The focus is not on disability but on the diversity of human ability.

Ability is thus like weight or height—it is positive-valued only. Nobody has dis-weight or dis-height; neither are there disabilities, only abilities. Any experience of disability is not attributable to a person but to a mismatch between a person's abilities and the ability assumptions of the environment. Like the proverbial water in a glass half full, abilities are only present and "designed for," not absent and "filled in."

This view of "design for" rather than "fill in" is not the historical view. Filling in for lost abilities has been the norm. From early human history through World War II and after, the approach has been to restore whatever was lost (such as an arm or a leg). People were expected to adapt themselves to the environment, whether physical or social, as they found it, with little hope that society would meet them halfway.

Although such attitudes have improved, designers and developers still often take a similar stance with interactive computing systems. When users' abilities fail to match the ability assumptions underlying today's interactive computing systems, the burden usually falls on the users to make themselves amenable to those systems, and the systems remain oblivious to the users doing it (see Figure 3).

Figure 3. Users adapting themselves to the ability assumptions of their input devices—keyboards and trackballs—which are oblivious to their contortions.


(a)

(b)

Ability and Situation

The experience of disability applies to us all. With the proliferation of smartphones, tablets, and wearables, we increasingly interact with systems in situations that challenge our abilities.

Consider how the physical environment of "the computer user" has changed from the 1980s to today. A typical computer user in the 1980s would have been seated at a stable work surface with ample lighting, controlled temperatures, quiet surroundings, and relatively few distractions. Today, with computing pervading so many aspects of life, "computer users" interact off-the-desktop while adapting to dynamic, distracting environments and their movements through them.[7] An example is how users interact in "four-second bursts"[24] when walking with smartphones, constantly diverting their attention from and returning to their screens. And yet, with the exception of a few research prototypes (such as in Mariakakis et al.[19]), smartphones are oblivious to users' behaviors, unchanging from the street to the café to the library to the office.

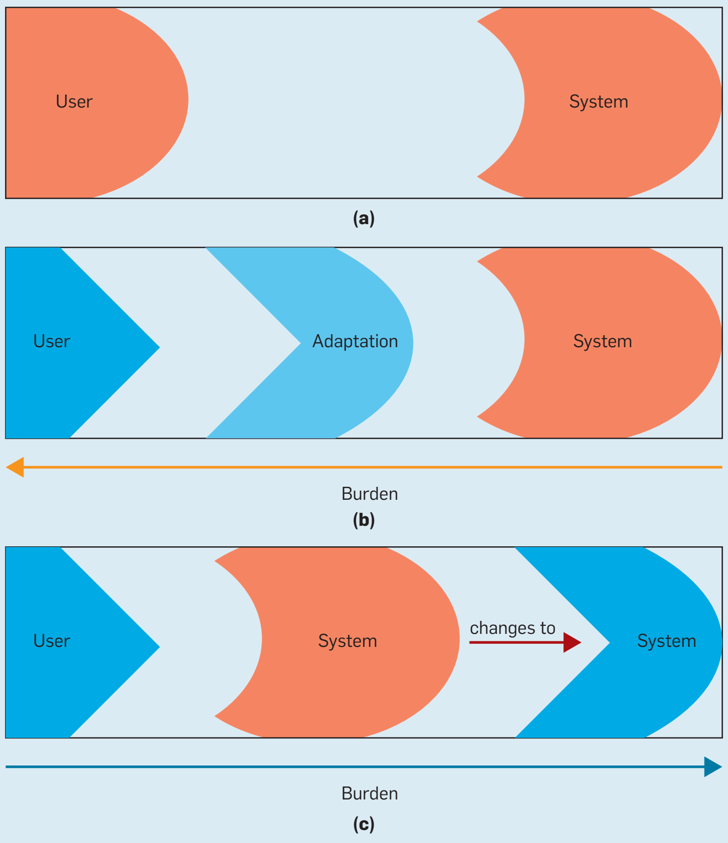
Researchers have identified "situational impairments" caused by changing situations, contexts, and environments, using the language of disability and accessibility.[7,22,27,33,38] Sears and Young[27] said, "Both the environment in which an individual is working and the current context . . . can contribute to the existence of impairments, disabilities, and handicaps."

This observation has grown even more relevant in the 15 years since it was made. In Stockholm, Sweden, city officials have erected street signs alerting drivers to watch out for people texting while walking. In Seoul, South Korea, some sidewalks are divided into two lanes, one for those intent on walking while staring at their phones, and the other for those who promise to refrain. In the U.S., the Utah transit authority imposed a $50

Table 1. Situational factors that can limit our physical and cognitive abilities and affect our interactions with technology.

Vibration	Cold temperatures
Divided attention	Impeding clothing (such as gloves)
Distraction	Encumbering baggage
Diverted gaze	Rainwater
Device out-of-view	Light levels (such as darkness and glare)
Intervening objects	Ambient noise
Bodily motion (such as walking)	Social behaviors (such as interruptions)
Vehicular motion	Multitasking
Uneven terrain	Stress
Physical obstacles	Fatigue
Awkward postures or grips	Haste
Occupied hands	Intoxication

Figure 4. User abilities and a system's ability assumptions: (a) user abilities match a system's ability assumptions; (b) in assistive technology, the user acquires an adaptation to remedy a mismatch; and (c) in ability-based design, user abilities drive changes in the system.



fine for "distracted walking," including walking while texting. And the city of Honolulu adopted the Distracted Walking Law, banning even just looking at a screen while in a crosswalk. Alarmingly, the Federal Communications Commission estimates that at any daytime moment in the U.S., 660,000 people are interacting with their smartphones while driving.[c]

If we are to design for human ability, disabling situations must be addressed. Unfortunately, our interactive computing systems know little about their users' abilities, attention, situations, contexts, and environments. A great many factors can impair use (see Table 1), yet few of them are detected, accommodated, or used as a basis for discouraging or deferring interaction.

Toward Ability-Based Design

Addressing such concerns while providing a unified approach to designing for people of all abilities is why we pursued ability-based design,[37,38] a design approach in which the human abilities required to use a technology in a given context are scrutinized, and systems are made operable by or adaptable to alternative abilities. Emerging from our work on adaptive user interfaces,[9–11] ability-based design is characterized by the designer's focus on what people can do, rather than on what they cannot do, and on systems and environments adapting to users rather than the other way around. Examples include desktop interfaces that customize their designs based on how a user moves a mouse,[10] touch surfaces that observe complex motor-impaired touch sequences and resolve intended touch points,[21] and mobile touch keyboards that sense and accommodate walking to improve accuracy.[12]

Strategies

Ability-based design is pragmatic, concerned with abilities insofar as they are useful for design. It is thus strategy-agnostic, embracing multiple methods for achieving successful user-technology fits. Strategies include automatic ability-based adaptation; high configurability by the end user; ability-specific customization by a third party; and having multiple designs for alternative abilities. Regardless of which one is employed, ability-based systems do the work to match users' abilities, not burdening users with having to satisfy a system's rigid ability assumptions.

Employing a visual language developed by Edwards,[3] we outline a successful user-system fit in Figure 4a, where a user's abilities match a system's ability assumptions. In traditional assistive technology, when they do not match, as in Figure 4b, the burden falls on the user to become amenable to the system by procuring an adaptation. The adaptation fits and makes the user "seem normal" to the system. With ability-based design, this burden is reversed

c https://www.fcc.gov/consumers/guides/dangers-texting-while-driving (see Figure 4c); it is the user's abilities that dictate what the system must do to make itself amenable to the user. For example, the system might adapt or be adapted to match the user's abilities.

Ability-based design differs from traditional assistive technology by eschewing user-procured adaptations like the one in Figure 4b in favor of on-board adaptability. When on-board adaptability is not possible or practical, assistive technologies can still meet the objectives of ability-based design if they are well matched to the user's abilities and not burdensome to procure. In cases where assistive technologies are used, ability-based systems should be aware of their use and do whatever they can to make that use as uninhibited as possible.

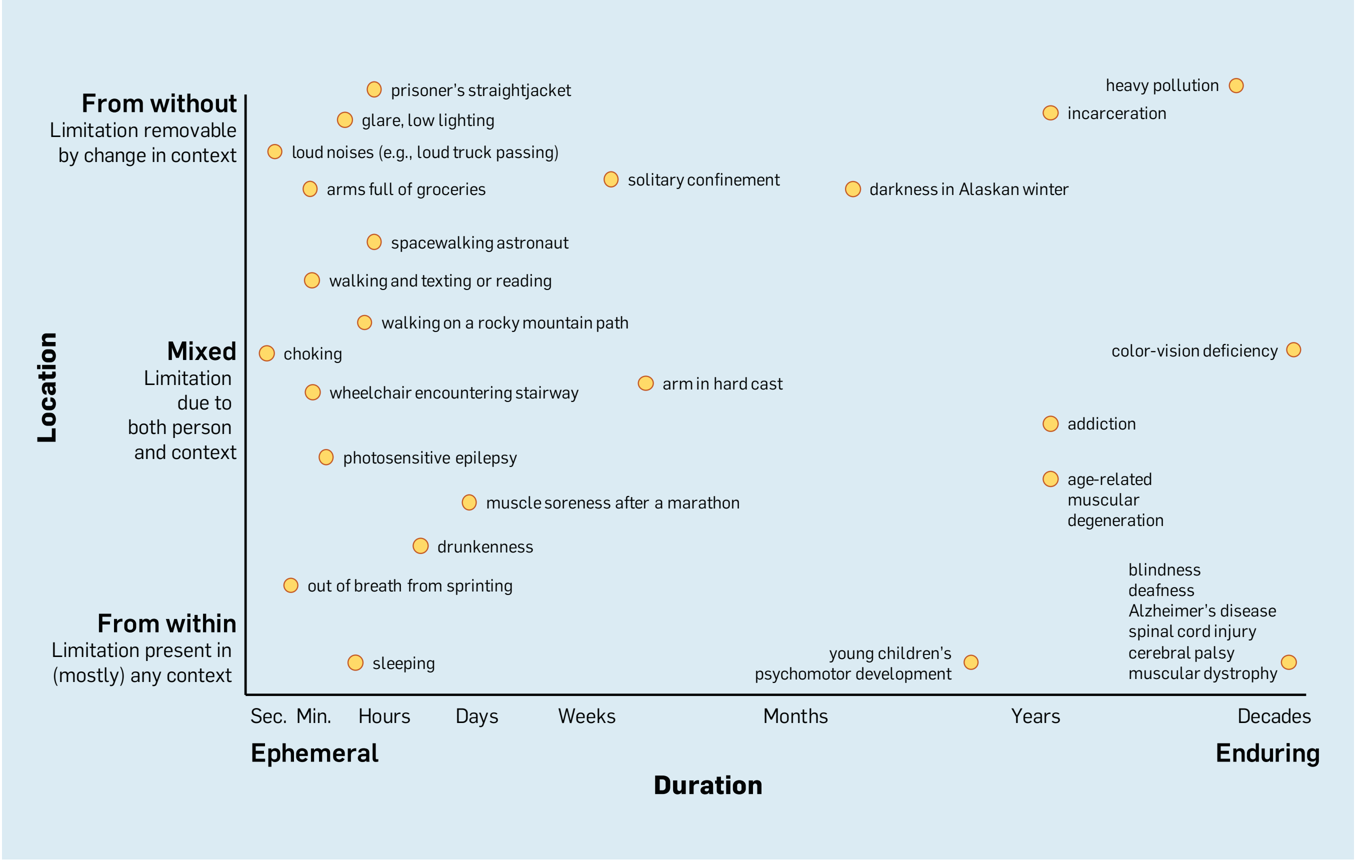
Ability-based design also relates to universal design.[18] Arising from the field of architecture, universal design readily applies to built structures and spaces and has been extended to physical and digital products as well. Universal design is the process of designing places and things so they are usable by people with the greatest range of abilities possible. Ability-based design creates designs that match the abilities of individual users to the greatest extent possible. Ability-based design is thus one way to realize the ambitions of universal design. Unlike universal design, however, we created ability-based design with interactive computing in mind, so sensing, adapting, and configuring are presumed technology possibilities. While ability-based design might not natively apply to immutable concrete stairs, as in Figure 1, it would thus ask how future stairways (or wheelchairs) might use sensing, adapting, and configuring to prevent accessibility barriers.

Other strategies for designing for diverse abilities exist and are similar to ability-based design insofar as they consider users' abilities and the role of the environment. For example, inclusive design[16,23] seeks to eliminate design choices that cause exclusion by revealing designer biases through participatory methods, field observations, and empathy building. Among the foci of inclusive design is understanding user capabilities, similar to ability-based design.

A key difference between ability-based design and both universal design and inclusive design is one of focus and approach. Universal design and inclusive design focus on creating designs that are for general widespread use, including by people with specific interface needs. Ability-based design promotes creating general interfaces with the flexibility to address a range of users, as well as tailored interfaces specific to subgroups or even to an individual user. Ability-based design potentially has broader reach since it embraces both flexible-general and tailored-specific interfaces in its scope and approach.

With ability-based design, there is also a subtle but important difference in focus by the researcher, designer, or developer. With universal design or inclusive design, the focus is on creating an interface that can accommodate as many people as possible. With ability-based design, the focus is on the abilities of the individual user. All three approaches might at times produce similar designs, but with ability-based design, the focus is on optimizing the

Figure 5. Contexts that impair one's ability to use technology are defined by location and duration. What advances in sensing and computing might enable systems to better serve their users across a range of contexts?



experience for individual users according to their abilities and contexts.

Contexts Limiting Technology Use

Ability-based design considers a broad range of contexts that impair technology use. We define a space with two axes: location and duration (see Figure 5). The location of a limitation ranges "from within the self" to "from outside the self." Limitations arising from within the self are present in almost any context. Examples are a spinal cord injury, a toddler's undeveloped psychomotor control, and being asleep. Changing a person's context has little effect on the limitations arising from such internal states.

In contrast, limitations arising from outside the self are present primarily due to context, and therefore changeable. Astronauts have remarkable physical abilities, but while spacewalking, expressing many of those abilities is quite difficult. Even an Olympic athlete can do little when confined to a prisoner's straightjacket. The external context severely limits the person's expressible abilities.

Intermediate points also exist on the location axis, where the mixture of self and environment limit ability. One example of a mixed-location limitation is photosensitive epilepsy, where a flashing light might induce seizures. If not for the flashing light, seizures would not be triggered. In this example, a part of the person and a part of the environment combine to pose a possible limitation.

On the other axis, the duration of a limitation ranges from "ephemeral" to "enduring." An ephemeral limitation lasts only briefly and changes quickly; one example is the lack of a usable arm because a person is carrying an infant. Next, short-term limitations can arise from many causes, including inebriation, illness, and an ankle sprain. Limitations might even be enduring or even lifelong, as with, say, those caused by age-related declines, spinal cord injuries, incurable diseases, lifetime imprisonment, or irreversible brain damage.

Our argument is not that the lived experience of a person with one arm is the same as that of a person carrying an infant. Situational impairments are neither subjectively nor objectively anything like long-term limitations. Rather, the argument is that technology designs that are useful to people with certain long-term limitations might also be useful to people in certain disabling situations. A technology design for a person with one arm also might be useful for a person carrying an infant. Using an ability-based lens helps one recognize such design opportunities.

Assistive technology focuses mainly on compensating for long-term limitations within a person, as in Figure 5, bottom right. Ability-based design considers a larger space of limitations that impair technology use.

Design Principles

By adopting ability-based design in numerous projects, we have formulated and refined seven design principles to guide our work (see Table 2). The first three are required of any ability-based design project and relate to the designer's attitude and approach, or "stance." The next two relate to adaptive or adaptable user interfaces, and the final two to sensing and modeling users and contexts. Taken together, they can help guide designers and developers creating ability-based systems.

Example Projects

Our development of ability-based design was and continues to be highly iterative and inductive, arising from research projects that both preceded and followed its initial formulation. Here, we highlight a number of projects to illustrate the possibilities for ability-based design:

SUPPLE. SUPPLE[9–11] was an automatic user-interface generator that used decision-theoretic optimization to help choose interface widgets and layouts that were optimized for a user's preferences, visual abilities, and motor abilities. For optimizing motor performance, SUPPLE first presented the user with a series of basic pointing, clicking, dragging, and list-selection tasks.[10] It then built regression models capturing the relationship between task parameters and user performance, using these models to guide the optimization process such that the interface being generated was predicted to be the fastest to operate by the user. Each user thus received a custom user interface, optimized for that user's particular abilities.

In a quantitative study in 2008 involving people with motor impairments,[11] SUPPLE's custom interfaces were 26% faster and 73% more accurate to use than the default interfaces provided by manufacturers of popular desktop software applications. SUPPLE thus helped close more than 60% of the performance gap between people with and people without motor impairments, making access more equitable. Qualitatively, it was apparent how SUPPLE was optimizing interfaces based on different abilities; for example, SUPPLE gave people with muscular dystrophy interfaces with small, densely packed targets able to support slow, short, deliberate movements. In contrast, SUPPLE gave people with cerebral palsy interfaces with large, spread-out targets divided among different tabs, compatible with fast but error-prone movements. SUPPLE had no declarative knowledge of either muscular dystrophy or cerebral palsy, generating its user interfaces solely from observed input performance.

The SUPPLE approach was used in subsequent projects. For example, in SPRWeb,[6] SUPPLE's personalized optimization approach was used to recolor websites, adapting them to the individual color-vision abilities of users with color-vision deficiencies. SPRWeb also aided users in color-limiting or color-altering situations, including glare and low-light conditions.

SUPPLE exhibited the first six principles of ability-based design and was the original system that inspired many of the ideas now found throughout ability-based design.

Slide Rule. Slide Rule[14] was a mobile screen reader that made touchscreens accessible to blind users by leveraging multi-touch gestures and audio feedback. It was an example of making systems usable to people with abilities different from what device manufacturers originally intended. Slide Rule addressed a pressing challenge emerging in 2007 from the advent of touchscreen smartphones: How would a blind person interact with a phone having buttons that person could not feel? At the time, smartphones had little or no accessibility support, and many people presumed touchscreens could not be made usable for blind people. Slide Rule developed a set of gestures and the first finger-driven screen-reading techniques to enable blind people to access and control smartphone touchscreens.

We became aware from a personal communication in 2010 that Slide Rule inspired aspects of Apple's VoiceOver screen reader for iOS. Indeed, Slide Rule's finger-driven screen reading, swipe gestures, and second-finger tap can all be found in VoiceOver today.

Slide Rule exhibited the first three principles of ability-based design; it also exhibited the fourth and sixth principles, as its screen reader could adapt to the speed of users' movements, tailoring its performance to theirs. The underlying principles demonstrated in Slide Rule have survived into today's touchscreen systems.

Walking user interfaces. Today's smartphones are portable but not truly mobile because they support interaction only poorly while moving; for example, walking divides attention,[24] reduces accuracy,[17] slows reading speed,[26] and impairs obstacle avoidance.[32] We conducted multiple projects to improve interaction while walking, focusing on people's abilities while on the go.

In our early exploration of walking user interfaces,[15] we studied level-of-detail (LoD) adaptations, where the interface shown while a user was standing had high detail and the interface shown while a user was walking had low detail, with larger fonts and bigger targets. When a user moved from standing to walking and vice versa, the interface changed. We compared this adaptive interface to component static interfaces for both walking and standing, finding that walking increased task time for static interfaces by 18%, but with our adaptive interface, walking did not increase task time. We also found that the adaptive interface performed like its component static interfaces; that is, there was no penalty for the LoD adaptation.

In our subsequent project, called WalkType,[12] we made mobile touch-based keyboards almost 50% more accurate and 12% faster while walking. Touch-based features like finger location, duration, and travel were combined with accelerometer features like signal amplitude and phase to train decision trees that reclassified wayward key-presses. WalkType effectively remedied a systematic inward rotation of the thumbs caused by whichever foot was moving forward as the user walked.

Performing input tasks is only one challenge while walking. Consuming output is another. In SwitchBack,[19] an attention-aware system for smartphones, a smartphone's front-facing camera was used to track eye-gaze position on the screen to aid task resumption. For example, when a user was reading and looked away, SwitchBack remembered the last-read line of text; when the user's gaze returned to the screen, that same line was highlighted to draw the user's attention for easy task resumption.

These three walking user interfaces exhibited all seven principles of ability-based design to varying degrees.

Table 2. Seven principles of ability-based design, updated and revised from previous versions.[37,38]

	Principle	***Description***
Designer Stance (required)	Ability	Designers focus on users' abilities, not disabilities, striving to leverage all that users can do in a given situation, context, or environment.
	Accountability	Designers respond to poor usability by changing systems, not users, leaving users as they are.
	Availability	Designers use affordable and available software, hardware, or other components acquirable through accessible means.
Adaptive or Adaptable Interface (optional)	Adaptability	Interfaces might be adaptive or adaptable to provide the best possible match to users' abilities.
	Transparency	Interfaces might give users awareness of adaptive behaviors and what governs them and the means to inspect, override, discard, revert, store, retrieve, preview, alter, or test those behaviors.
Sensing and Modeling (optional)	Performance	Systems might sense, monitor, measure, model, display, predict, or otherwise utilize users' performance to provide the best possible match between systems and users' abilities.
	Context	Systems might sense, monitor, measure, model, display, predict, or otherwise utilize users' situation, context, or environment to anticipate and accommodate effects on users' abilities.

Global Public Inclusive Infrastructure

Ability-based design has been applied mostly at the level of individual systems and applications, but for greater impact, a new infrastructure that extends beyond the user's own device is needed. Although the Global Public Inclusive Infrastructure (GPII),[34,35] with its cloud-based auto-personalization of information and communication technologies, was formulated independent of ability-based design, its objectives are the same—enable interfaces to be ideally configured to match each user's situated abilities.

The GPII is built on three technological pillars.[35] The second, "auto-personalization," is the one of interest here.[d] Its long-term goal is to ensure that any digital interface a person encounters instantly changes to a form that can be understood and used by that person. The GPII's auto-personalization capability uses a person's needs and preferences, which are stored in the cloud or on a token, to automatically configure the interface of each device for that individual.[34,36] Its "one size fits one" approach is designed to help each person have the "best fit" interface possible. Since interface flexibility on current devices and software is limited, GPII auto-personalization uses both built-in features and assistive technologies (AT) (on the device and in the cloud)

d The two other pillars make it easy for people to determine what they need or prefer, ensuring solutions exist for everyone.

to achieve each best-fit interface. For example, accessibility features located in five layers—operating system features, installed AT, browser features, cloud AT, and Web app features—can be configured to work together to provide best-fit user interfaces, with features at each level being invoked (or not) in order to meet the user's needs and preferences.

GPII auto-personalization supports interfaces that self-adapt, as well as configuration of interfaces and adaptations, to match a user's needs. By combining auto-adjusting interfaces, preference-configured interfaces, and user-selected-and-configured AT, the GPII can function as a bridge among these approaches, maximizing the utility of each one for an individual at any point in time. The GPII also supports auto-configuration based on contextual changes.[40] The GPII thus meets all seven principles of ability-based design.

Taking Up the Challenge

Pursuing these and other projects, some patterns have emerged for us. For example, we noticed a perspective shift as we began to actively seek out the abilities people have, inspiring an openness to consider how we could create or change technologies to suit different abilities. We also noticed a seamlessness between designing for people with limited abilities and designing for people in ability-limiting situations. We realized accessibility is indeed a worthy goal for all users. Because we were looking to modify systems, not users, we deemphasized assistive hardware add-ons. Customization arose from a powerful sequence of sensing, modeling, and adapting; it also arose from support for end-user configurability, as with the U.S. Air Force cockpits mentioned earlier. We thus made our interactive systems more aware of their users and contexts.

Where does ability-based design go next? One way to answer is to treat the vision of ability-based design as a grand challenge and ask what it would take to create a world in which anyone, anywhere, at any time could interact with technologies that are ideally suited to his or her situated abilities. Achieving the "anyone anywhere any time" part will require systemwide infrastructure of the kind pursued by the GPII. Ability-aware operating systems infused with SUPPLE-like user-interface generators could help create personalized applications. Improved sensing and modeling of users' abilities and contexts, as in walking user interfaces, could enable mobile and wearable systems to better support diverse contexts of use. One challenge is to avoid explicit task-based training and calibration in favor of implicit observation and modeling from everyday use, as in Evans and Wobbrock[5] and Gajos et al.[8]

What would it take to create a world in which anyone, anywhere, at any time could interact with technologies that are ideally suited to his or her situated abilities?

To date, ability-based design has focused primarily on single-user experiences, but the social lives of users could also lend themselves to collaborative support. How should the abilities of a pair, group, team, crowd, or organization be considered? For service arrangements, what would it look like to have an ability-based design for services?

Moreover, abilities exist on many levels, from low-level sensorimotor and cognitive abilities, to mid-level abilities for daily living, to high-level social, occupational, professional, and creative abilities. Such abilities form a hierarchy paralleling Maslow's hierarchy of needs,[20] whereby each need corresponds to an ability to meet it. Ability-based design seems applicable throughout such a hierarchy, but the range has yet to be explored.

Concerning "adaptivity," providing each individual with a unique user interface raises several pragmatic issues, as in, say, authoring help documentation, provision of customer support, and making the design process of personalized experiences consistent with accepted design practice. These challenges are real but, as we discuss elsewhere,[9] solvable.

With the vast range of human abilities from which to draw, adaptivity based on sensing and modeling is a powerful way to realize custom designs that, while inevitably imperfect, nonetheless provide good user-system fits at scale. Adaptive interfaces can remember users' abilities and preferences and draw on them when generating interfaces for both familiar and unfamiliar systems, providing more satisfying and effective access for each individual user. We thus see an important and continuing role for adaptivity and personalization within ability-based design.

We close with a quote from Frank Bowe (1947–2007), professor and disability-rights activist who helped instigate the Americans with Disabilities Act of 1990 (https://www.ada.gov/). Writing in *MIT Technology Review* in 1987, he emphasized the importance of focusing on what people are able to do, not on what holds people back:[1] "When society makes a commitment to making new technologies accessible to everyone, the focus will no longer be on what people cannot do, but rather on what skills and interests they bring to their work. That will be as it always should have been."

We could not agree more.

Acknowledgments

We wish to thank our co-authors on the projects we covered here, especially Jeffrey Bigham, Leah Findlater, Jon Froehlich, Mayank Goel, Susumu Harada, Alex Mariakakis, Shwetak Patel, and Daniel S. Weld. This work was supported in part by the Mani Charitable Foundation and the National Science Foundation under grants IIS-0952786 and CNS-1539179. Any opinions, findings, conclusions or recommendations expressed in this work are those of the authors and do not necessarily reflect those of any supporter or collaborator. C

References

1. Bowe, F. Making computers accessible to disabled people. *MIT Technology Review 90* (Jan. 1987), 52–59, 72.
2. Dourish, P. *Where the Action Is: The Foundations of Embodied Interaction.* MIT Press, Boston, MA, 2001.
3. Edwards, A.D.N. Computers and people with disabilities. Chapter 2 in *Extra-Ordinary Human-Computer Interaction: Interfaces for Users with Disabilities*, A.D.N. Edwards, Ed. Cambridge University Press, Cambridge, England, 1995, 19–43.
4. Engelbart, D.C. and English, W.K. A research center for augmenting human intellect. In *Proceedings of the AFIPS Fall Joint Computer Conference* (San Francisco, CA, Dec. 9–11). AFIPS, Los Alamitos, CA, 1968, 395–410.
5. Evans, A.C. and Wobbrock, J.O. Taming wild behavior: The input observer for obtaining text entry and mouse pointing measures from everyday computer use. In *Proceedings of CHI 2012* (Austin, TX, May 5–10). ACM Press, New York, 2012, 1947–1956.
6. Flatla, D.R., Reinecke, K., Gutwin, C., and Gajos, K.Z. SPRWeb: Preserving subjective responses to website colour schemes through automatic recolouring. In *Proceedings of CHI 2013* (Paris, France, Apr. 27–May 2). ACM Press, New York, 2013, 2069–2078.
7. Gajos, K.Z., Hurst, A., and Findlater, L. Personalized dynamic accessibility. *Interactions 19*, 2 (Mar.-Apr. 2012), 69–73.
8. Gajos, K.Z., Reinecke, K., and Herrmann, C. Accurate measurements of pointing performance from in situ observations. In *Proceedings of CHI 2012* (Austin, TX, May 5–10). ACM Press, New York, 2012, 3157–3166.
9. Gajos, K.Z., Weld, D.S., and Wobbrock, J.O. Automatically generating personalized user interfaces with SUPPLE. *Artificial Intelligence 174*, 12–13 (Aug. 2010), 910–950.
10. Gajos, K.Z., Wobbrock, J.O., and Weld, D.S. Automatically generating user interfaces adapted to users' motor and vision capabilities. In *Proceedings of UIST 2007* (Newport, RI, Oct. 7–10). ACM Press, New York, 2007, 231–240.
11. Gajos, K.Z., Wobbrock, J.O., and Weld, D.S. Improving the performance of motor-impaired users with automatically generated, ability-based interfaces. In *Proceedings of CHI 2008* (Florence, Italy, Apr. 5–10). ACM Press, New York, 2008, 1257–1266.
12. Goel, M., Findlater, L., and Wobbrock, J.O. WalkType: Using accelerometer data to accommodate situational impairments in mobile touch-screen text entry. In *Proceedings of CHI 2012* (Austin, TX, May 5–10). ACM Press, New York, 2012, 2687–2696.
13. Hummels, C. and Lévy, P. Matter of transformation: Designing an alternative tomorrow inspired by phenomenology. *Interactions 20*, 6 (Nov.-Dec. 2013), 42–49.
14. Kane, S.K., Bigham, J.P., and Wobbrock, J.O. Slide Rule: Making mobile touch screens accessible to blind people using multi-touch interaction techniques. In *Proceedings of ASSETS 2008* (Halifax, Nova Scotia, Canada, Oct. 13–15). ACM Press, New York, 2008, 73–80.
15. Kane, S.K., Wobbrock, J.O., and Smith, I.E. Getting off the treadmill: Evaluating walking user interfaces for mobile devices in public spaces. In *Proceedings of MobileHCI 2008* (Amsterdam, the Netherlands, Sept. 2–5). ACM Press, New York, 2008, 109–118.
16. Keates, S., Clarkson, P.J., Harrison, L.-A., and Robinson, P. Towards a practical inclusive design approach. In *Proceedings of CUU 2000* (Arlington, VA, Nov. 16–17). ACM Press, New York, 2000, 45–52.
17. Lin, M., Goldman, R., Price, K.J., Sears, A., and Jacko, J. How do people tap when walking? An empirical investigation of nomadic data entry. *International Journal of Human-Computer Studies 65*, 9 (Sept. 2007), 759–769.
18. Mace, R.L., Hardie, G.J., and Place, J.P. Accessible environments: Toward universal design. Chapter 8 in *Design Intervention: Toward a More Humane Architecture*, W.E. Preiser, J.C. Vischer, and E.T. White, Eds. Van Nostrand Reinhold, New York, 1991, 155–176.
19. Mariakakis, A., Goel, M., Aumi, M.T.I., Patel, S.N., and Wobbrock, J.O. SwitchBack: Using focus and saccade tracking to guide users' attention for mobile task resumption. In *Proceedings of CHI 2015* (Seoul, South Korea, Apr. 18–23). ACM Press, New York, 2015, 2953–2962.
20. Maslow, A.H. A theory of human motivation. *Psychological Review 50*, 4 (July 1943), 370–396.
21. Mott, M.E., Vatavu, R.-D., Kane, S.K., and Wobbrock, J.O. Smart touch: Improving touch accuracy for people with motor impairments with template matching. In *Proceedings of CHI 2016* (San Jose, CA, May 7–12). ACM Press, New York, 2016, 1934–1946.
22. Newell, A.F. Extra-ordinary human-computer interaction. Chapter 1 in *Extra-Ordinary Human-Computer Interaction: Interfaces for Users with Disabilities*, A.D.N. Edwards, Ed. Cambridge University Press, Cambridge, England, 1995, 3–18.
23. Newell, A.F. and Gregor, P. User-sensitive inclusive design: In search of a new paradigm. In *Proceedings of CUU 2000* (Arlington, VA, Nov. 16–17). ACM Press, New York, 2000, 39–44.
24. Oulasvirta, A., Tamminen, S., Roto, V., and Kuorelahti, J. Interaction in 4-second bursts: The fragmented nature of attentional resources in mobile HCI. In *Proceedings of CHI 2005* (Portland, OR, Apr. 2–7). ACM Press, New York, 2005, 919–928.
25. Rose, L.T. *The End of Average: How We Succeed in a World That Values Sameness.* HarperCollins, New York, 2015.
26. Schildbach, B. and Rukzio, E. Investigating selection and reading performance on a mobile phone while walking. In *Proceedings of MobileHCI 2010* (Lisbon, Portugal, Sept. 7–10). ACM Press, New York, 2010, 93–102.
27. Sears, A. and Young, M. Physical disabilities and computing technologies: An analysis of impairments. Chapter 25 in *The Human-Computer Interaction Handbook, First Edition*, J.A. Jacko and A. Sears, Eds. Lawrence Erlbaum, Hillsdale, NJ, 2003, 482–503.
28. Shinohara, K. and Wobbrock, J.O. In the shadow of misperception: Assistive technology use and social interactions. In *Proceedings of CHI 2011* (Vancouver, BC, Canada, May 7–12). ACM Press, New York, 2011, 705–714.
29. Shinohara, K. and Wobbrock, J.O. Self-conscious or self-confident? A diary study conceptualizing the social accessibility of assistive technology. *ACM Transactions on Accessible Computing 8*, 2 (Jan. 2016), article 5.
30. Stienstra, J. Embodying phenomenology in interaction design research. *Interactions 22*, 1 (Jan.-Feb. 2015), 20–21.
31. Sutherland, I.E. Sketchpad: A man-machine graphical communication system. In *Proceedings of the AFIPS Spring Joint Computer Conference* (Detroit, MI, May 21–23). AFIPS, Santa Monica, CA, 1963, 329–346.
32. Vadas, K., Patel, N., Lyons, K., Starner, T., and Jacko, J. Reading on-the-go: A comparison of audio and handheld displays. In *Proceedings of the Eighth Conference on Human-Computer Interaction with Mobile Devices and Services* (Helsinki, Finland, Sept. 12–15). ACM Press, New York, 2006, 219–226.
33. Vanderheiden, G.C. Anywhere, anytime (+anyone) access to the next-generation WWW. *Computer Networks and ISDN Systems 29*, 8–13 (Sept. 1997), 1439–1446.
34. Vanderheiden, G. and Treviranus, J. Creating a global public-inclusive infrastructure. In *Proceedings of the International Conference on Universal Access in Human-Computer Interaction, Lecture Notes in Computer Science, Vol. 6765* (Orlando, FL, July 9–14). Springer, Berlin, Germany, 2011, 517–526.
35. Vanderheiden, G., Treviranus, J., Ortega-Moral, M., Peissner, M., and de Lera, E. Creating a global public-inclusive infrastructure (GPII). In *Proceedings of the International Conference on Universal Access in Human-Computer Interaction, Lecture Notes in Computer Science, Vol. 8516* (Heraklion, Crete, Greece, June 22–27). Springer, Berlin, Germany, 2014, 506–515.
36. Vanderheiden, G.C., Treviranus, J., Usero, J.A.M., Bekiaris, E., Gemou, M., and Chourasia, A.O. Auto-personalization: Theory, practice and cross-platform implementation. In *Proceedings of the Human Factors and Ergonomics Society Annual Meeting* (Boston, MA, Oct. 22–26). Human Factors and Ergonomics Society, Santa Monica, CA, 2012, 926–930.
37. Wobbrock, J.O. Improving pointing in graphical user interfaces for people with motor impairments through ability-based design. Chapter 8 in *Assistive Technologies and Computer Access for Motor Disabilities*, G. Kouroupetroglou, Ed. IGI Global, Hershey, PA, 2014, 206–253.
38. Wobbrock, J.O., Kane, S.K., Gajos, K.Z., Harada, S., and Froehlich, J. Ability-based design: Concept, principles, and examples. *ACM Transactions on Accessible Computing 3*, 3 (Apr. 2011), article 9.
39. World Health Organization. *Document A29/INFDOCI/1*, 1976, Geneva, Switzerland.
40. Zimmermann, G., Vanderheiden, G.C. and Strobbe, C. Towards deep adaptivity: A framework for the development of fully context-sensitive user interfaces. In *Proceedings of the International Conference on Universal Access in Human-Computer Interaction, Lecture Notes in Computer Science, Vol. 8513* (Heraklion, Crete, Greece, June 22–27). Springer, Berlin, Germany, 2014, 299–310.

Jacob O. Wobbrock (wobbrock@uw.edu) is a professor of human-computer interaction in the Information School and, by courtesy, in the Paul G. Allen School of Computer Science & Engineering at the University of Washington, Seattle, WA. USA.

Krzysztof Z. Gajos (kgajos@eecs.harvard.edu) is a Gordon McKay Professor of Computer Science at the Harvard Paulson School of Engineering and Applied Sciences at Harvard University, Cambridge, MA, USA.

Shaun K. Kane (shaun.kane@colorado.edu) is an assistant professor in the Department of Computer Science and, by courtesy, in the Department of Information Science, at the University of Colorado Boulder, Boulder, CO, USA.

Gregg C. Vanderheiden (GreggVan@umd.edu) is a professor and Director of the Trace R&D Center in the College of Information Studies at the University of Maryland College Park, MD, USA, and Co-Director of the Global Public Inclusive Infrastructure.

DOI:10.1145/3209086

Text analysis can reveal patterns of association among medical terms and medical codes.

BY DAVID GEFEN, JAKE MILLER, JOHNATHON KYLE ARMSTRONG, FRANCES H. CORNELIUS, NOREEN ROBERTSON, AARON SMITH-MCLALLEN, AND JENNIFER A. TAYLOR

Identifying Patterns in Medical Records through Latent Semantic Analysis

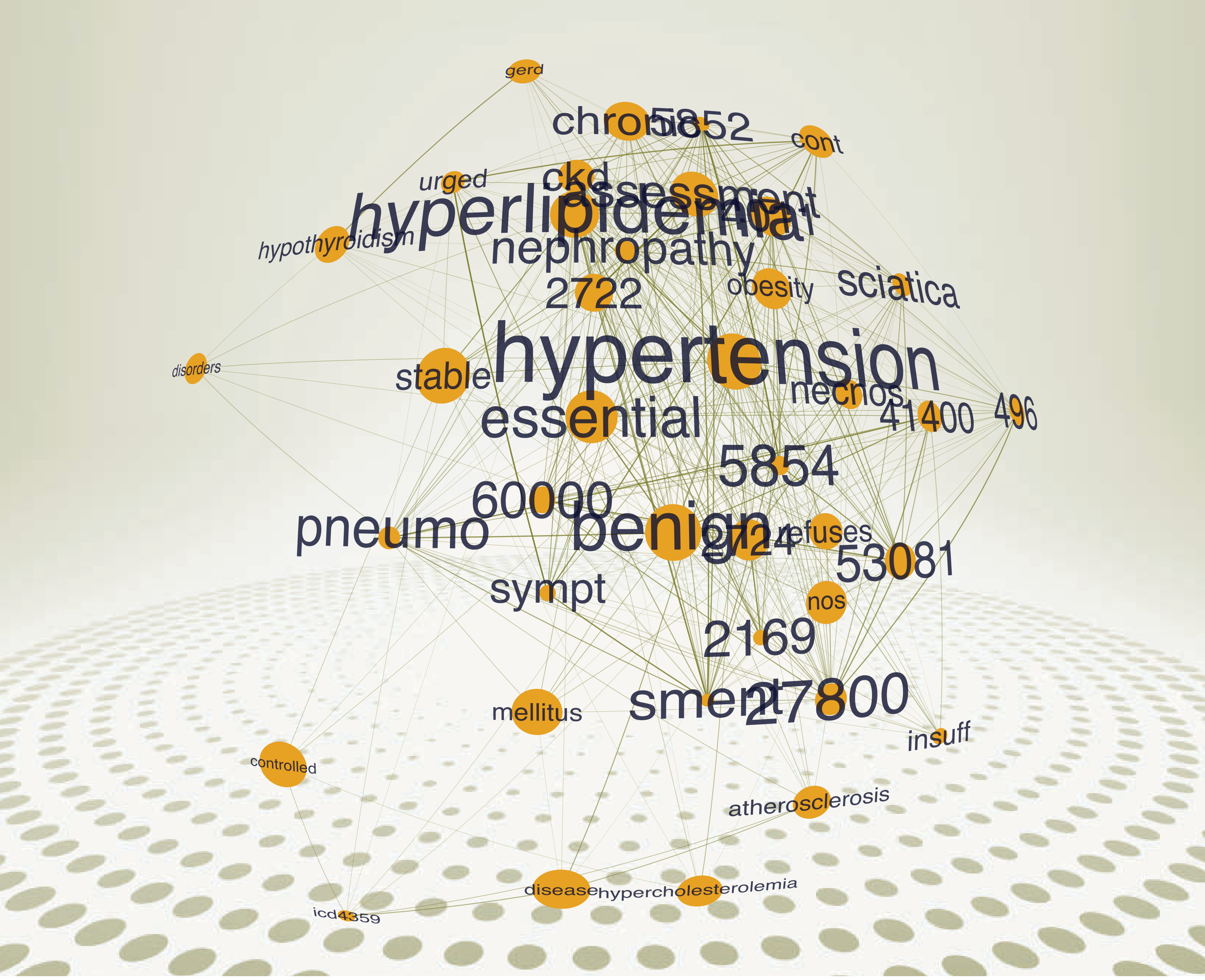
MOUNTAINS OF DATA are constantly being accumulated, including in the form of medical records of doctor visits and treatments. The question is what actionable information can be gleaned from it beyond a one-time record of a specific medical examination. Arguably, if one were to combine the data in a large corpus of many patients suffering from the same condition, then overall patterns that apply beyond a specific instance of a specific doctor visit might be observed. Such patterns might reveal how medical conditions are related to one another over a broad set of patients, as well as how these conditions might be related to the International Classification of Diseases, 10th Revision, Clinical Modification (ICD-10-CM) of the Centers for Disease Control and Prevention (CDC) Classification of Diseases, Functioning, and Disability codes (henceforth, ICD codes[a]). Conceivably, applying such a method to a large dataset could even suggest new avenues of medical and public health research by identifying new associations, along with the relative strength of the associations compared to other associations. It might even be applied to identify possible side effects in phase IV drug testing. Moreover, that potential might be even more potent if it could also identify indirect, or through other terms, connections among terms.

To address such medical-analysis objectives, this article explores in a preliminary manner the applicability of a method based on latent semantic analysis (LSA) that transforms the term-document [frequency] matrix (TDM) through a singular value decomposition (SVD) into a semantic space. Once such a semantic space is created, the lexical distance among terms and among combinations of terms can be calculated by projecting them onto that space. More important, lexical distance can be calculated even when terms are associated only indirectly; that is, when the terms do not appear together in any one shared document but are related to one another through another set of terms;[10,12] see Gefen et

a ftp://ftp.cdc.gov/pub/Health_Statistics/NCHS/Publications/ICD10CM/2018/

» key insights

- **Analyzing electronic health records through latent semantic analysis shows that medical terms and keywords known to be related to the same medical condition co-appear frequently across documents.**
- **This allows identifying a typical pattern of medical terms and keywords as they relate to a specific medical condition, along with the strength of their relative association to that condition.**
- **It also allows identifying previously unknown medical terms and keywords that may be related to that medical condition.**



Word cloud of data.

al.[4] and Holzinger et al.[8] for discussions of other text-analysis methods.

The study demonstrates how LSA can be applied to sanitized medical records dealing with congestive heart failure to identify patterns of association among terms of interest, as well as among those terms and the ICD codes that appear for the same patient. By "sanitized," researchers mean documents after removing protected health information (PHI) content. The data was provided by Independence Blue Cross (IBX), a health insurer, and deals with congestive heart failure.

Some associations revealed through LSA in this study were expected (such as the one between hypertension and obesity). Associations might be obvious, but identifying them is essential because it shows the credibility of the method. Other associations were less expected by medical experts (such as the infrequent association LSA identified between "hypertension" and "sciatica"). That association might indicate a one-person issue, highlighting the potential for identifying associations among medical terms through LSA that might reveal cases that require special attention or unexpected, possibly previously unknown, relationships. As we explain in the next section, which describes LSA, associations identified by LSA might also include terms that do not appear together in any document but rather are associated with one another through their joint association to another term.

Past research that applied LSA to medical science showed LSA can identify shared ontologies across scientific papers, even if terms have different names,[15] and the degree that concepts are shared across papers in the *Proceedings of the National Academy of the Sciences* can reveal expected patterns.[13] The study adds a new angle to the accumulating literature on LSA in medical contexts by showing its potential contribution to medical science by associating medical terms and ICD codes as applied in practice in medical reports, especially by adding an ordinal scale of how close the terms are to one another compared to other terms. For example, the cosines we use[b] suggest that in this population hypertension is closer to being benign than chronic and even

b https://github.com/jakemiller3/GefenEt-CACM-MedicalLSA/blob/master/Gefen%20et%20al%20Online%20Appendix.pdf less related to hypothyroidism. The results also suggest the method could be applied to assist in the management of medical treatment by identifying unusual cases for special attention.

Introduction to LSA

LSA creates a semantic space from which it is possible to derive lexical closeness information; that is, how close terms or documents are to one another in a corpus. LSA creates that space by first creating a TDM from a relevant corpus of documents and then running an SVD on that TDM. The TDM is a frequency matrix that records how often each term appears in each document. Before the TDM is created, the text in the corpus is often stemmed and stop words are excluded. Stop words are words that occur frequently (such as "the" and "or") and thus add little or no semantic information to the documents or to how terms relate to one another. There are default lists of stop words in English and other languages in R and other software packages. Additional words of interest can be added to these lists so they, too, are excluded from the semantic space. It is also common in LSA practice to remove accents, cast the text in lower case, and remove punctuation marks.

After the TDM is created researchers often apply a process of weighting, whereby the frequency numbers are replaced with a transformation that considers the distribution of each term in the document it appears in and across the documents in the corpus. Researchers typically apply both local and global weighting. Local weighting gives more weight to terms that appear more often in a single document. Global weighting gives less weight to terms that appear in many documents. One of the most common weighting transformations is the term "frequency-inverse document frequency," or TF-IDF, transformation. Some research (such as by Beel et al.[1]) claims that TF-IDF is the most common text-mining transformation, giving more weight to terms that appear often in a given document but less weight if the terms appear frequently in the corpus as a whole. It is also a recommended type of transformation.[14] Stemming, stop-word removal, weighting transformation, and other preparatory steps are standard options in R packages that are used to create a semantic space; R is a free and popular statistical language.

LSA reveals not only that terms are related but also the degree of that relationship compared to other relationships as a set of ordinal cosine distance measures.

Once the semantic space is created, researchers can project terms and combinations of terms onto it; likewise, documents and parts of documents can be projected on it to produce various measures of semantic closeness.[2,12] That degree of semantic closeness is typically a cosine value that provides an ordinal scale of relatedness of terms and documents. Terms and documents that are close to one another in meaning also have higher degrees of closeness, as revealed by LSA.[6] This allows researchers to use LSA to identify synonyms.[6,9,16] In fact, LSA has been used so successfully in identifying such closeness levels that it has been shown to answer introduction-to-psychology multiple-choice exam questions almost as well as students do[12] and score on the Test of English as a Foreign Language exam almost as high as nonnative speakers.[11] LSA can also classify articles into core research topics.[3] The semantic space created by LSA can be so realistic that LSA has even been applied to identifying how questionnaire items factor together.[5]

Our study has sought to show that applying standard LSA packages in R is enough to produce associations among medical terms and ICD codes in electronic health records covering medical visits, including their relative strength in comparison to other associations, as corroborated by subject-matter experts, and do so even when applying the packages with only their default settings and without transforming the data beyond the automated transformation the packages introduce. Applying the standard transformation is important because it is not practical for researchers to manually correct typos, alternative spellings, shorthand, or optical character recognition (OCR) errors. Processing such "dirty" data without manual correction is necessary in real real-world applications.

Data

The data we used was provided by IBX in a joint research project with Drexel University. It consisted of 32,124 text files obtained by running an OCR on the medical transcripts of 1,009 scanned medical charts of 416 distinct patients who suffered or were suspect-ed of suffering from congestive heart failure in 2013 and 2014. IBX removed associated patient identifiers, demographics, and cost data. The IBX Privacy office and Drexel University medical institutional review board (IRB) both approved the research protocol in advance. The medical records consisted of the text portions of the medical record in one file and the ICD medical codes in another file. An artificial patient ID key replaced the actual patient ID in each medical report and each ICD list of codes. The medical reports were combined by that patient's ID.

We analyzed the data as is. We did not correct the data for alternative equivalent spellings (such as "catheterization" and "catheterisation"). Nor did we correct the data for obvious spelling mistakes and OCR errors (such as correcting "cardioverterdefibrillator" to "cardioverter-defibrillator"). This was done deliberately so the power of LSA could be shown even when run on untreated raw data. This was important because manually correcting medical reports is both costly and prone to introducing additional error. Manually checking these words revealed them to be mostly misspellings.

Analysis

We created the TDM after all the words were cast as lower case and punctuations and the standard set of stop words removed. Numbers were not removed from the raw data as they could have represented ICD codes. We then subjected the TDM to a TF-IDF transformation before a SVD was run on it, retaining 100 dimensions. There are no standard rules of thumb for how many dimensions to retain because dimensionality depends on context and corpora.[13] Adding more SVD dimensions inevitably results in more nuance and variance, as well as more noise.

Knowing the data concerned congestive heart failure, we identified the closest neighbor terms to "cardiac" and "hypertension" after creating the semantic space. The cosine distances are listed in the github.com link mentioned earlier, omitting terms that appeared in fewer than four patient records, as well as in strings with 10 or more numeric digits (such as phone numbers). Figure 1 and Figure 2 show the heat-map clustering for the terms "hypertension" and "cardiac," respectively, and Figure 3 outlines the LSA process. It is important to emphasize that LSA also reveals indirect associations among terms (such as when one term is related to another only through a third term), a key advantage of LSA over manual inspection.

A researcher might correctly associate terms that appear together but could miss those related only indirectly. That is beside the obvious advantage of LSA in that the analysis can be done semi-automatically and on very large corpora quickly and might otherwise require an unrealistic investment of time if done manually. Moreover, and crucially important, LSA reveals not only that terms are related but also the degree of that relationship compared to other relationships as a set of ordinal cosine distance measures. These distances can be used for other analyses (such as clustering terms to determine structures within the text or to compare documents). The ability to run other analyses could be particularly helpful for identifying connections that heretofore had not been identified and thus can be used as a predictive model to enable disease screening, early detection, and intervention.

Demonstrating this ability to identify relationships, the cosine distances in the github.com link show that the term "hypertension" is most closely related to "hyperlipidemia," which, according to the American Heart Association, means "high levels of fat particles (lipids) in the blood." Considering that this is a very common condition, estimated to afflict 31 million Americans, it might be expected. The terms "benign" and "essential" are also close, as is the diagnostic code "4011," or "icd4011," which, in the ICD9 code, means "benign essential hypertension." Hypertension is also, as expected, closely related to "obesity" and "mellitus" (diabetes), hardening arteries ("atherosclerosis"), acid reflux ("gerd"), and high cholesterol ("hypercholesterolemia"). However, hypertension is also seman-

Figure 1. Clustering of the 40 terms and ICD-9-CM codes closest to "hypertension."

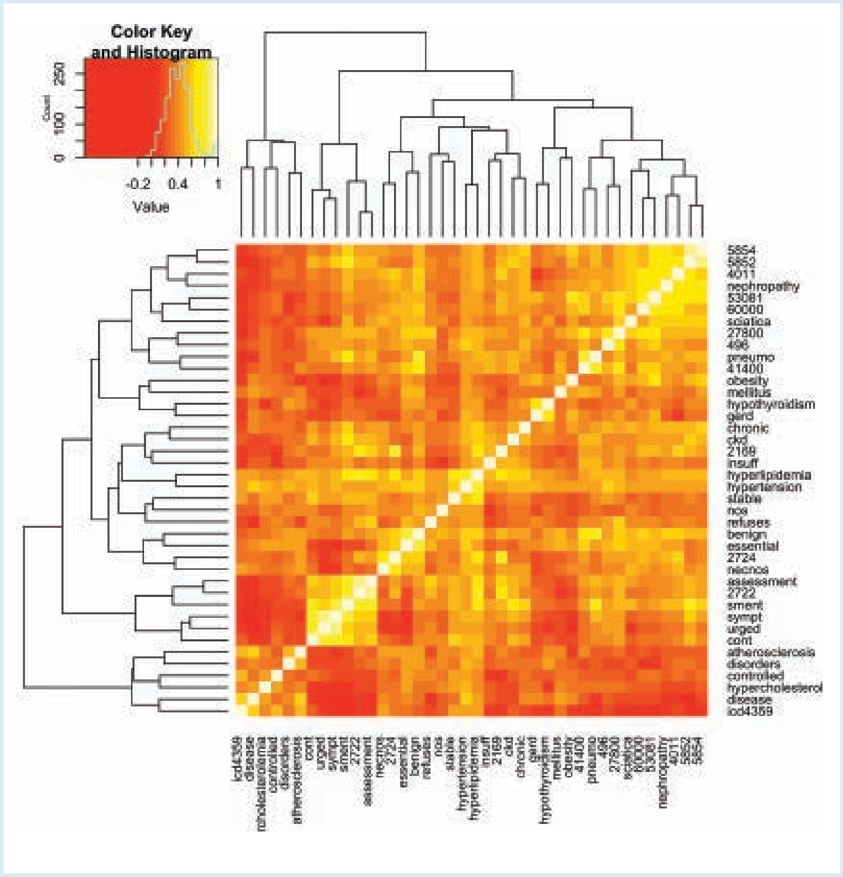


Figure 2. Clustering of the 40 terms and ICD-9-CM codes closest to "cardiac."

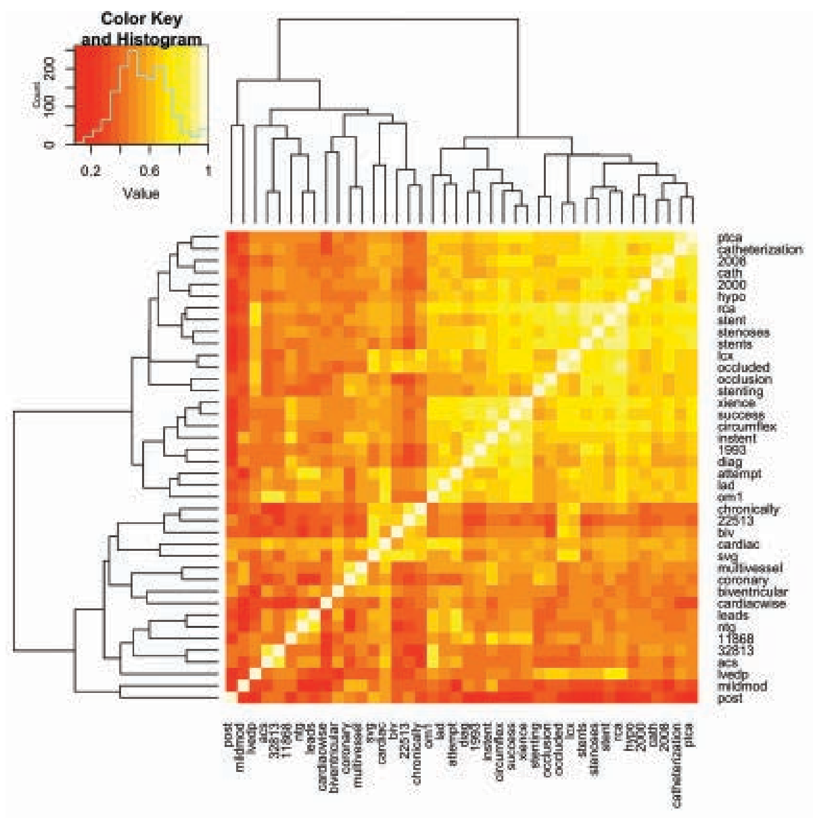
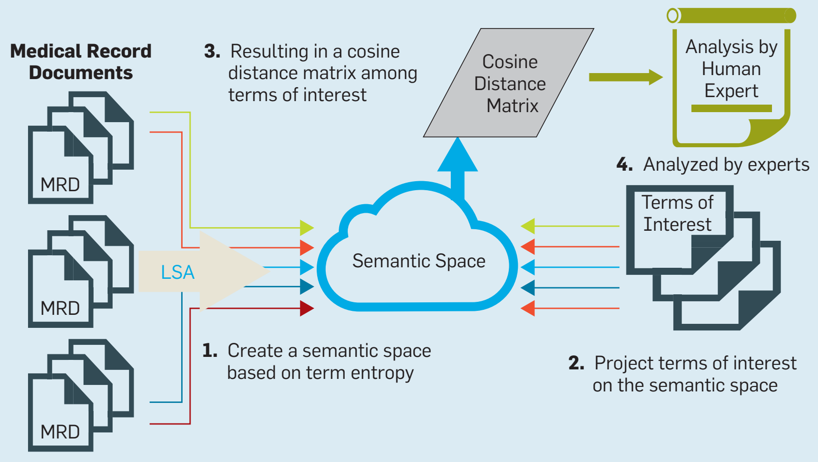


Figure 3. Outline of the latent semantic analysis process.


Medical Record
Documents
MRD
MRD
MRD
LSA
3. Resulting in a cosine distance matrix among terms of interest
Cosine
Distance
Matrix
Analysis by
Human
Expert
4. Analyzed by experts
Semantic Space
Terms of
Interest
1. Create a semantic space based on term entropy
2. Project terms of interest on the semantic space

tically close to other terms and diagnostic codes (such as "chronic airway obstruction," or "496"), kidney disease ("ckd," nephropathy, 5852, and 5854), prostate issues ("prostatic" and code "60000"), and others. The terms and codes associated with hypertension in this population are more related to cholesterol- and fat-related terms than to kidney disorders, based on cosine distances. Our analysis also identified relationships to "sciatica" and "cerebral ischemia" ("icd4359") with restricted blood flow in the brain. The heat map in Figure 1 of cosine distances between terms shows how these terms relate to one another. For example, "urged," "sympt," and "assessment," relate to patient interactions, closely associated with diagnosed ("2722") mixed hyperlipidemia at the lower left of the Figure. Similarly, several kidney-related terms and codes are clustered at the upper right. Corroboratively, the analysis tied "hypertension" and "gastroesophageal reflux disease (GERD)" with 91 unique cases, two seemingly unrelated health conditions that only as of March 2017 were found to be related.[17]

The cosines we provide show "cardiac" connecting most strongly to the first obtuse marginal (OM1) and left anterior descending (LAD) arteries, as well as to the saphenous vein graft ("SVG") procedure. Perhaps because it is an adjective, and not a condition, "cardiac" is associated with terms pertaining to body parts and procedures more than diagnoses. The heat map in Figure 2 shows a noticeable cluster of terms relating to catheterization and stents (such as "stent," "stenting," "xience," and "instent"). Clustering identifies measurements of cardiac performance as well, with ventricular activation time ("vat") and left ventricular end-diastolic pressure ("lvedp"). This emphasis, as identified through clustering, is further reflected when compared with the cosine distances of "hypertension" (see footnote b). The nearest terms in the heat map to "cardiac" have greater cosine values, indicating a smaller distance relative to the neighbors of "hypertension." Such close association implies that "cardiac" has a more focused meaning in our texts, whereas "hypertension" is associated with a larger range of disparate terms, in this case, frequently co-occurring conditions and diagnoses. This may also be because cardiac issues are often acute, with specific actions rendered as treatments, while hypertension is more a chronic disease, associated with many related diseases.

Discussion

The analysis in our study dealt with a relatively small PHI-cleansed sample of medical-records data from a well-defined context. We caution against drawing medical conclusions from such a sample, but the results are indicative of the potential in applying LSA to such contexts. We created a semantic model that identified known relationships among medical terms, relating diagnoses and treatments. We scrubbed the sample of most demographic data; the dataset was too small anyway to allow cross-sectional analysis. Constructing a model from a larger, more detailed dataset could yield substantial potential for medical discovery. Comparing reports across patients could provide even more information, as by, say, enabling creation of a "typical" profile of care/treatment trajectory, as well as diagnosis and prognosis as they apply to disease, condition names, or ICD codes. Such a profile could conceivably lead to early detection and allow identifying exceptional cases in need of immediate medical attention.

A typical profile for a condition or ICD code could help create a method to at least partly support phase IV testing of new drugs involving long-term monitoring of the effects of drugs following approval by the U.S. Food & Drug Administration. LSA could improve this process not only by automating it but also by identifying a drug's possible indirect effects, or the effects associated with the drug but only through other diagnosis. So, for example, if drug A is associated with condition B and condition B is associated with condition C, then LSA will identify that A and C might be related. A human examiner might not notice it but could be aided by LSA to identify possible connections of interest for the expert to consider; see Holzinger et al.[7] for more on interactive machine learning.

Analyzing medical records could also allow comparison of diagnosis and prognosis across populations (such as differentiating between men and women). Accounting for demographics could also indicate the prevalence of diagnosis and prognosis by age and by geographical area, possibly indicating hazardous environmental conditions. Moreover, given the diversity within society, running LSA on medical records could also allow quasi-experimental design studies, as in, say, comparing clinics in areas where unique treatments are allowed against those where they are not. Planning such an experiment would be difficult and IRB approval might not always be forthcoming, but if the treatment conditions occur in the population, then studying them would not be so contentious and could become routine. LSA could support such an after-the-effect examination.

Combining LSA cosine distances with TDM frequency values may also allow identification of extraordinary cases that are closely related to a condition but very rare. In the data we had, we omitted terms that appeared in fewer than four records, but the relationships between rare diseases and more common ones might suggest new avenues of research for a range of health issues. As a supplement to medical practice, a text-analytic approach might be able to suggest alternative diagnoses based on documented symptoms that might otherwise be attributed to more common conditions.

Above all, a key advantage of LSA is that it allows rank ordering of related terms. Being able to assign numbers, and hence categorize how much a condition is related to other conditions, could provide insight about identifying what symptoms, and how much more than others, they might indicate a problem (such as hypertension).

Conclusion

In a musical parody, lyricist David Lazar wrote in a song called "Dr. Freud" that Sigmund Freud's disciples said, "... by God, there's gold in them thar ills." There certainly was. Maybe there also is in gleaning medical insight from medical records documents through LSA. C

References

1. Beel, J., Gipp, B., Langer, S., and Breitinger, C. Research-paper recommender systems: A literature survey. *International Journal on Digital Libraries 17*, 4 (Nov. 2016), 305–338.
2. Deerwester, S., Dumais, S.T., Furnas, G.W., Landauer, T.K., and Harshman, R. Indexing by latent semantic analysis. *Journal of the American Society for Information Science 41*, 6 (1990), 391–407.
3. Evangelopoulos, N., Zhang, X., and Prybutok, V.R. Latent semantic analysis: Five methodological recommendations. *European Journal of Information Systems 21*, 1 (Jan. 2012), 70–86.
4. Gefen, D., Endicott, J., Fresneda, J., Miller, J., and Larsen, K.R. A guide to text analysis with latent semantic analysis in R with annotated code studying online reviews and the Stack Exchange community. *Communications of the Association for Information Systems 41*, 21 (Dec. 2017), 450–496.
5. Gefen, D. and Larsen, K. Controlling for lexical closeness in survey research: A demonstration on the technology acceptance model. *Journal of the Association for Information Systems 18*, 10 (Oct. 2017), 727–757.
6. Gomez, J.C., Boiy, E., and Moens, M.-F. Highly discriminative statistical features for email classification. *Knowledge Information Systems 31*, 1 (Apr. 2012), 23–53.
7. Holzinger, A., Plass, M., Holzinger, K., Crisan, G.C., Pintea, C.-M., and Palade, V. A glass-box interactive machine learning approach for solving NP-hard problems with the human-in-the-loop; https://arxiv.org/abs/1708.01104
8. Holzinger, A., Schantl, J., Schroettner, M., Seifert, C., and Verspoor, K. Biomedical text mining: State-of-the-art, open problems and future challenges. Chapter in *Interactive Knowledge Discovery and Data Mining Biomedical Informatics, Lecture Notes in Computer Science LNCS 8401*, A. Holzinger and I. Jurisica, Eds. Springer, Berlin, Heidelberg, Germany, 2014, 271–300.
9. Islam, A., Milios, E., and Keselj, V. Text similarity using Google tri-grams. In *Proceedings of the 25th Canadian Conference on Artificial Intelligence* (Toronto, Canada, May 28–30). Springer, Toronto, Canada, 2012, 312–317.
10. Kintsch, W. Predication. *Cognitive Science 25*, 2 (2001), 173–202.
11. Landauer, T.K. and Dumais, S.T. A solution to Plato's problem: The latent semantic analysis theory of acquisition, induction, and representation of knowledge. *Psychological Review 104*, 2 (1997), 211–240.
12. Landauer, T.K., Foltz, P.W., and Laham, D. An introduction to latent semantic analysis. *Discourse Processes 25*, 2 and 3 (1998), 259–284.
13. Landauer, T.K., Laham, D., and Derr, M. From paragraph to graph: Latent semantic analysis for information visualization. *Proceedings of the National Academy of Sciences 101*, 1 (Apr. 6, 2004), 5214–5219.
14. Larsen, K.R. and Bong, C.H. A tool for addressing construct identity in literature reviews and meta-analyses. *MIS Quarterly 40*, 3 (Sept. 2016), 529–551.
15. Larsen, K.R., Michie, S., Hekler, E.B., Gibson, B., Spruijt-Metz, D., Ahern, D., Cole-Lewis, H., Ellis, R.J.B., Hesse, B., Moser, R.P., and Yi, J. Behavior change interventions: The potential of ontologies for advancing science and practice. *Journal of Behavioral Medicine 40*, 1 (Feb. 2017), 6–22.
16. Valle-Lisboa, J.C. and Mizraji, E. The uncovering of hidden structures by latent semantic analysis. *Information Sciences 177*, 19 (Oct. 2007), 4122–4147.
17. Zhiwei, H., Meiping, C., Jimin, W., Qing, S., Chao, Y., Xing, D., and Zhonggao, W. Improved control of hypertension following laparoscopic fundoplication for gastroesophageal reflux disease. *Frontiers of Medicine 11*, 1 (Mar. 2017), 68–73.

David Gefen (gefend@drexel.edu) is a professor in the Decision Sciences and MIS Department, Academic Director of the Doctorate in Business Administration Program, and Provost Distinguished Research Professor in the LeBow College of Business at Drexel University, Philadelphia, PA, USA.

Jake Miller (jlm479@drexel.edu) is an assistant clinical professor in the Management Department in the LeBow College of Business at Drexel University, Philadelphia, PA, USA.

Johnathon Kyle Armstrong (Kyle.Armstrong@ibx.com) is a research scientist at Independence Blue Cross, Philadelphia, PA, USA.

Frances H. Cornelius (fc28@drexel.edu) is a professor in the College of Nursing and Health Professions and Chair of the MSN Advanced Role Department, Complementary and Integrative Health Department, and coordinator of Clinical Nursing Informatics Education in the College of Nursing and Health Professions at Drexel University, Philadelphia, PA, USA.

Noreen Robertson (nmr26@drexel.edu) is the Associate Vice Dean for research at Drexel University College of Medicine and a research assistant professor in the Department of Biochemistry & Molecular Biology at Drexel University, Philadelphia, PA, USA.

Aaron Smith-McLallen (aaron.smith-mclallen@ibx.com) is the Director of Data Science and Health Care Analytics at Independence Blue Cross, Philadelphia, PA, USA.

Jennifer A. Taylor (jat65@drexel.edu) is an associate professor of environmental and occupational health in the School of Public Health at Drexel University, Philadelphia, PA, USA.

This study was supported by Drexel Grant #282847.

DOI:10.1145/3132696

When it comes to anonymizing cryptocurrencies, one size most definitely does not fit all.

BY DANIEL GENKIN, DIMITRIOS PAPADOPOULOS, AND CHARALAMPOS PAPAMANTHOU

Privacy in Decentralized Cryptocurrencies

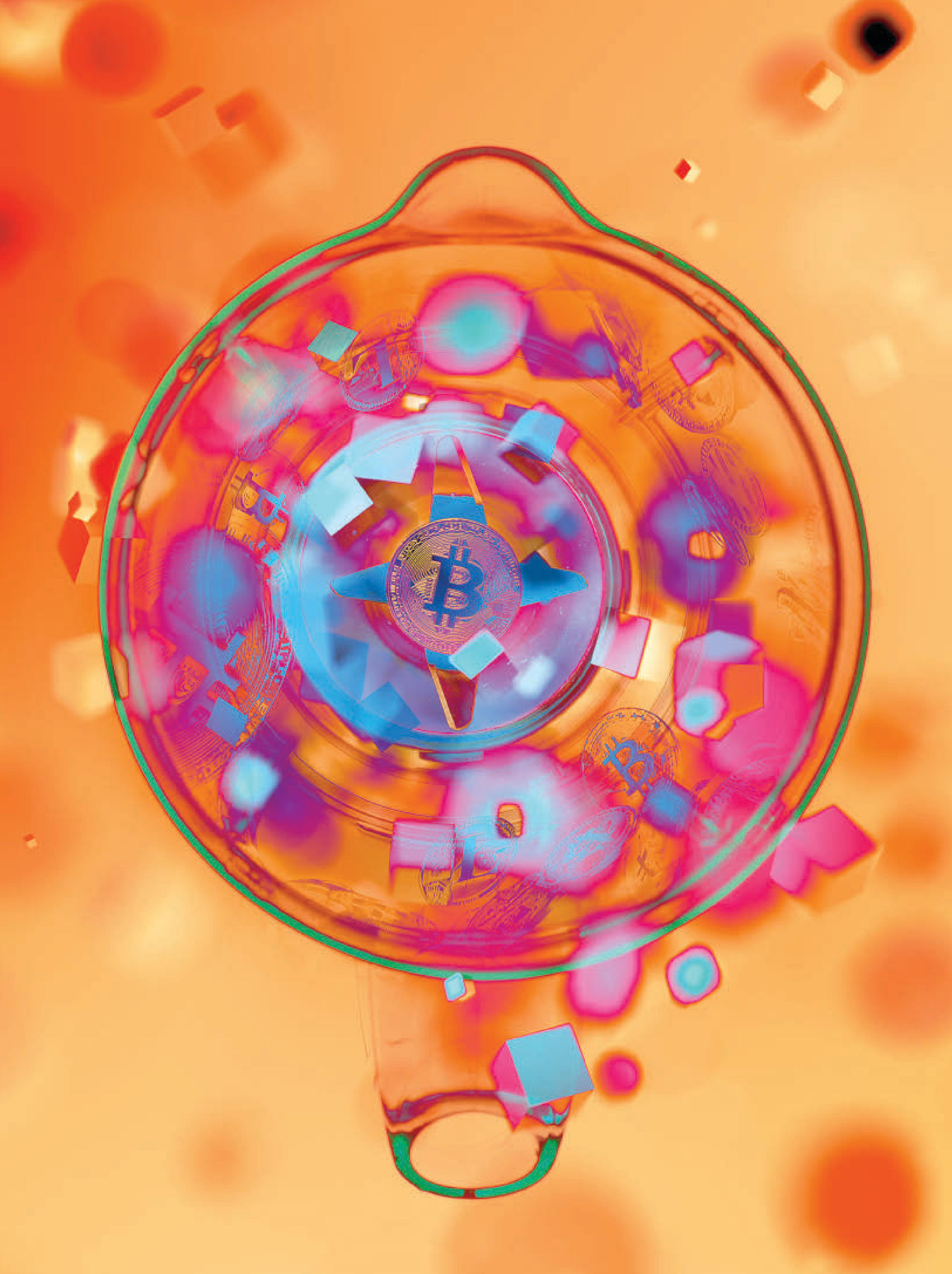
CRYPTOCURRENCIES PROMISE TO revolutionize the financial industry, forever changing the way we transfer money. Instead of relying on a central authority (for example, a government entity or a bank) to issue and manage money, cryptocurrencies rely on the mathematical design and security proofs of the underlying cryptographic protocols. Using cryptography and distributed algorithms, cryptocurrencies offer a fully decentralized setting where no single entity can monitor or block the transfer of funds. Cryptocurrencies have grown from early prototypes to a global phenomenon with millions of participating individuals and institutions.[17] Bitcoin[28] was the first such currency launched in 2009 and in the years since has grown to a market capitalization of over $15 billion (as of January 2017). This has led to the emergence of many alternative cryptocurrencies with additional services or different properties as well as to a fruitful line of academic research.

Apart from its other benefits (decentralized architecture, small transaction fees, among others), Bitcoin's design attempts to provide some level of "pseudonymity" by not directly publishing the identities of the participating parities. Every user interacts with the network by establishing a public address that acts as a "pseudonymous identity." In practice, there is no bound on the number of addresses a user can create; therefore there exists no single address a user can be related with. However, this pseudonymity is far from the desired *unlinkability* property in centralized e-cash protocols,[11] where when Alice sends an amount to Bob, the original source of these funds cannot be deduced. The reason for this problem is that in most decentralized cryptocurrencies all transaction information (payer and payee address, amount, among others) is publicly visible, stored in a distributed data structure called *blockchain* (for example, see www.blockchain.info). Therefore, an attacker can easily observe how money flows. This can lead to quite devastating deanomyization attacks and therefore there is a need for cryptocurrencies with stronger privacy guarantees.

In this article, we review widely studied mechanisms for achieving privacy in blockchain-based cryptocurrencies such as Bitcoin. We focus on mixing services that can be used as a privacy overlay on top of a cryptocurrency; and privacy-preserving alternative coins that,

» key insights

- **While blockchain-based cryptocurrencies like Bitcoin do not directly reveal users' identities, they are often prone to de-anonymization attacks. By observing the flow of transactions stored in the public blockchain, third parties can make accurate guesses about the identities of involved individuals.**
- **Existing privacy-enhancing techniques for cryptocurrencies mostly come in two flavors: Mixing overlay protocols that can be executed on top of an existing cryptocurrency to hide the flow of funds among a set of participants, and alternative privacy-preserving cryptocurrencies that use advanced cryptographic techniques to achieve strong user privacy by design.**
- **We review and compare solutions from both techniques.**



by design, aim to achieve strong privacy properties. We discuss and compare the privacy guarantees achieved by known mechanisms, as well as their performance and practical adoption.

Background: Bitcoin and Privacy

Bitcoin in a nutshell. The full mechanics of the Bitcoin protocol are rather involved and we refer interested readers to Bonneau et al.[7] and Nakamoto[28] (also see the "Inside Risks" column p. 20 in this issue). In the sequel, we provide a high-level abstraction of the protocol, highlighting the aspects that have the most impact on user privacy. A Bitcoin user participates in the protocol by first generating a cryptographic public/private key pair. The first operates as her public address: she can use it to send money to or receive money from other users in the same way one uses a bank account. Unlike a bank account though, a user can generate as many public/private key pairs as she wants—even one for every transaction. A simple transaction from user A to B contains a declaration of "A sends x bitcoins to B" signed with A's secret key.

These transactions are propagated via a flooding mechanism over an ad hoc, peer-to-peer network and are thus visible to every participant. Special users known as *miners* collect transactions and store them into blocks. These blocks are subsequently stored in a global public ledger of transactions known as blockchain. This chain is a sequential order of blocks, each of which references the previous one. Who appends the latest block is decided in a randomized manner using a proof-of-work mechanism that generally guarantees that the amount of blocks a miner gets to generate (receiving a corresponding miner's fee) is proportional to the ratio of its computational power over the total power of all miners in the protocol.

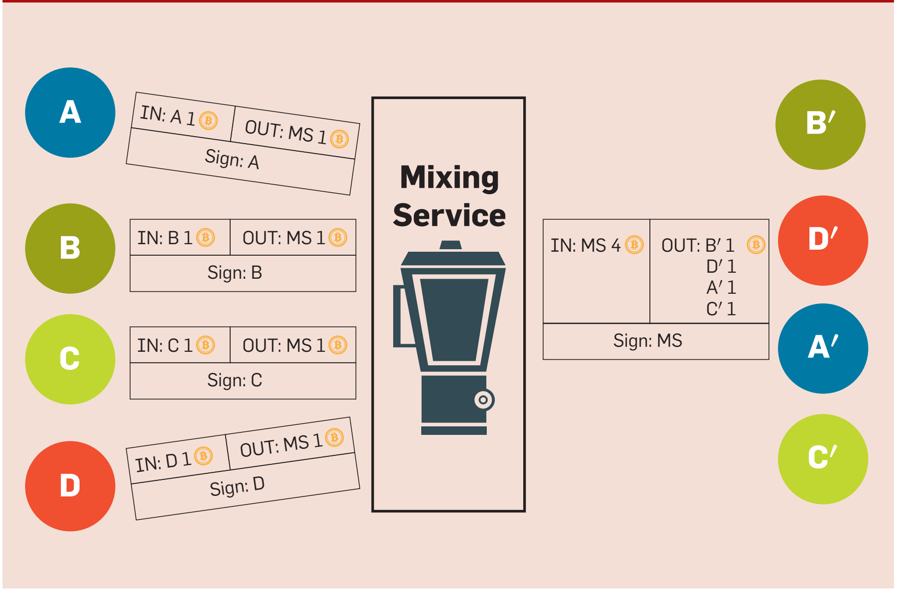
In order for a miner to add a transaction in the next block it must first be validated. Consider the case of the transaction from A to B. Before adding it to the current block, the miner must check that it is signed by A, and A did not previously spend these bitcoins. The former is easy to achieve given the public key of A (embedded in the transaction). The latter can be verified by tracing A's entire transaction history to check that the bitcoins in question where not previously spent (in practice, it suffices to just trace unspent transactions).

Deanonymization attacks. This scenario highlights a crucial issue regarding Bitcoin's privacy: Transaction validation is founded on public access to the transaction history. While the physical identity of the owner of address A (Alice) cannot be directly deduced from this, any observer can see that the same individual performed a given set of transactions. Even worse, although Alice can always create a new address A' she might have to transfer money from her old one to the new one in order to use it, at which point an external observer can make an educated guess that links A' to the owner of A. Indeed, there has been a growing amount of literature[1,23,30,31] showing how the transaction graph can be used to link together addresses. This, combined with external information (for example, vendor purchases) and further heuristic analysis, can lead to user identification. Currently, there even exist Bitcoin tracking companies (for example, Elliptic[a] and Chainalysis[b]) that monitor the body of transactions, aiming at identifying illicit activity.

Network-level deanonymization. By default, the Bitcoin peer-to-peer protocol does not protect the IP addresses of the participants since they are communicated in the clear. Researchers[4] have shown how this information can be used to deduce user identities. However, most Bitcoin client implementations can be configured to run over an anonymous Tor proxy, hiding the participants' addresses. Unlike what one might expect, this approach does not solve the problem. Subsequent work[5] has demonstrated how the interaction between Bitcoin and Tor can be exploited by an adversary who not only compromises user privacy (negating the anonymizing effect of the latter) but can also launch a stealthy man-in-the-middle attack, targeting the security of the Bitcoin protocol itself. While Biyukov[5] discusses a number of partial countermeasures, to the best of our knowledge there is no definitive way to protect the network-level anonymity of Bitcoin users yet. One likely candidate solution is the evolution of Bitcoin to operate over a tailor-made anonymity network that will not suffer from the issues discussed here. It should be noted that none of the techniques we mention in the sequel addresses network anonymity explicitly.

Real-world anonymity. We stress the gap between anonymity as a property of the cryptocurrency protocol execution and "real-world anonymity." For example, when one uses a cryptocurrency to purchase goods or services from a vendor they must provide the latter with certain personal information (identity for registration, physical address for delivery, email for pur-

Figure 1. Example of centralized mixing with four participants and a trusted mixer. No observer can "link" input to output addresses.



a https://www.elliptic.co
b https://www.chainalysis.com

chase confirmation, and so on). Thus, the vendor can trivially link the public key with its owner, in a strong sense. Moreover, this information may be extracted by others (for example, in case the vendor is hacked or a government agency issues a subpoena). Combined with "Know-your-Customer" anti-money laundering policies that enforce the collection of such data (like the one included in the USA Patriot Act of 2001) this can seriously compromise the privacy of cryptocurrency users.

A Bitcoin address can be potentially mapped to a physical entity by examining its related history of transactions that are stored on the publicly accessible blockchain.

Bitcoin Mixing

As discussed, a Bitcoin address can be potentially mapped to a physical entity by examining its related history of transactions (namely edges on the transaction graph) that are stored on the publicly accessible blockchain. This has prompted researchers to introduce various techniques for achieving anonymity.[22] One such prominent approach is *Bitcoin mixing* (or *Bitcoin tumbling*).

Suppose each one of the addresses A, B, C, and D wish to send one bitcoin to addresses A', B', C', and D' respectively. If these transactions are posted directly on the blockchain, everybody can deduce exactly how money flows. Bitcoin mixing "mixes" these transactions so the amount of information that becomes public is minimized—with Bitcoin mixing one would just find out that A's bitcoin went to one of A', B', C', or D', but not to which address exactly. The simplest way to achieve that is to use a trusted mixer (as we will discuss) who first receives the money from A, B, C, and D and then sends the money to A', B', C', and D' respectively. Clearly such an approach does not reveal information about the exact transaction edges. In order for this process to truly hide the link between input and output addresses, all users must participate with the same amount. (One can always use a larger amount and specify a fresh "change" address.) This provides privacy similar to k-anonymity[36] (assuming k participants) since no observer can distinguish which coins end up at each recipient.

Bitcoin mixing methods. There are various ways of Bitcoin mixing, achieving different levels of privacy, security, and efficiency. One key distinction has to do with how the parties that participate are coordinated. In theory, it is always possible for a party that wants to mix its coins to find a friend with similar goals and coordinate the exchange of some amount of bitcoins via an out-of-bound channel (for example, phone). This is a valid solution but in order to truly improve their privacy, users should try to hide inside a set of parties that is as large as possible. On the other hand, point-to-point coordination of hundreds or thousands of users can be very impractical, especially if the execution of the mixing protocol requires multiple rounds of communication. Therefore, many centralized solutions have been proposed where a third-party server, that receives a mixing fee, is utilized to handle the logistics of the transaction, under varying threat models (fully trusted, accountable, or untrusted). Finally, one must consider whether or not the identities of the mixing participants (or even the link between sender and recipient) will be revealed to other participants.

Centralized mixers. The simplest and easiest way to implement a form of Bitcoin mixing is via a trusted third party that serves as the *mixer* (shown in Figure 1). To send an amount of bitcoins from an address A to another address A', A first performs a transaction transferring a fixed amount to the mixer and sends an encryption of A' under the mixer's public key to the latter. After collecting a number of such transactions (assuming the same amount in each transaction) from multiple users—or, alternatively, after a certain amount of time has elapsed—the mixer sends, in a single Bitcoin transaction containing the recipients' addresses in a randomly permuted order, the same amount back to recipients' addresses. This achieves k-anonymity for a set that is as large as the number of parties that use the mixer within the given time increment, as there is no way for an external observer to distinguish the mapping between input and output addresses. The anonymity set can be further increased beyond the number of parties that use the mixer in the given time increment by sequentially mixing the coins multiple times (using several mix transactions), at the cost of reduced efficiency. One thing to note is this approach does not hide the fact these users used the mixer (and may, therefore, have "something to hide").

There exist multiple providers (for example, Bitmixer,[c] Bitlaunder,[d] Helix[e]) that offer this service for a small mixing fee with varying degrees of adoption. However, the most notable problem is that this approach requires "blindly trusting" the mixer. What if the mixer goes out of business? What if it is forced (for example, via a subpoena) to reveal the actual transaction links? Most importantly, what if it simply steals the coins? All these are valid issues and have indeed been, to some extent, observed in practice (for example, see Möser[27]).

Avoiding coin theft by the mixer. To mitigate the problem of coin theft by the mixer, Bonneau et al. proposed Mixcoin,[8] a Bitcoin mixer that holds the provider *accountable*. Theft is still possible but it can be reported via the use of signed *warrants*. In particular, before receiving A's coins, the mixer signs a statement of "*if A sends me x BTC by time t_1, I will send x′BTC back to B by time t_2*" (where x' is slightly smaller than x to account for a mixing fee) and sends this statement (with off-chain communication) to A. In case the mixer does not follow up on its end, A can publish this warrant damaging its reputation.

The first solution to truly avoid the possibility of coin theft was CoinSwap,[21] whose main building block is a timed-escrow protocol between two parties (also known as a 2-of-2 escrow). At a high level, a timed-escrow protocol that transfers money from Alice to Bob is implemented with the following transactions. The *initial* transaction is posted by Alice and places a number of bitcoins in escrow for a time window t. For Bob to claim these coins, a *release* transaction must be posted, signed by both Alice and Bob, before time t. Otherwise, the funds return to Alice. CoinSwap avoids coin theft by the mixer using two *correlated* timed-escrow protocols, one between the payer and the mixer and one between the mixer and the recipient, such that the recipient receives the money if and only if the mixer receives money from the sender. The downside of Coinswap is it requires multiple rounds of interaction and waiting for the validation of at least two blocks in the blockchain. Moreover, like Mixcoin, it also exposes the participants' identities to the mixer.

Hiding users' identities from the mixer. Blindcoin[37] is an extension to Mixcoin that utilizes *blind signatures* for the warrants. Blind signatures[11] operate like regular cryptographic signatures but allow a party to sign a message without knowing the message's exact content. A user A that wishes to mix her coins initially provides only a commitment to a fresh address she owns, "blinded" by a random value. After receiving the warrant from the mixer, A can remove the randomness and publish the address to a public log that contains all addresses to which the mixer must forward coins for that epoch. This completely hides the link between input and output address of a user even from the mixer itself, achieving full unlinkability. However, since Blindcoin builds on Mixcoin it can only offer a limited notion of security: a cheating mixer can steal a participant's coins, but the participant can prove that a theft took place thus damaging the mixer's reputation.

Achieving full security and unlinkability. A more recent proposal is TumbleBit,[16] which simultaneously achieves full unlinkability and avoids coin theft. TumbleBit merges techniques from secure two-party computation and zero-knowledge proofs in order to protect the user's privacy and the validity of the transaction (including enforcing the mixer to carry it out honestly). At the core of TumbleBit is the notion of an RSA *puzzle*. In order for a party A to anonymously send a number of bitcoins to party B (assuming B has just established a fresh public address) via a mixer M, the interaction proceeds in three phases as follows.

First, during an *escrow phase*, A posts an initial escrow transaction for a number of bitcoins to M and B contacts M requesting that M post a similar initial escrow transaction toward him. Assume that the signature that B needs from M in order to claim the escrowed value is σ. Moreover, B obtains from M (via an off-chain protocol execution) an RSA puzzle that consists of two values z, c. The former is $z = \varepsilon^e \bmod N$ for some ε where N, e is the public RSA key of M (that is, z is a deterministic RSA "encryption" of ε). Recall that without access to the corresponding RSA decryption key d, it is not possible for anyone to retrieve ε at this point. The key property is that the second part of the puzzle, c, is a symmetric key encryption of signature σ using ε as the key. That is, if B could decrypt the RSA encryption and retrieve ε, he would be able to use it to decrypt c, retrieve ε, and post the necessary release transaction to claim the bitcoins escrowed by M.

Then, during a payment phase, B will utilize A to get a solution for the puzzle. For this, B sends A a blinded version of z, by choosing randomness r and sending $z' = r^e\, z \bmod N$. Then A sends z' to M, asking him to provide a solution ε' for this version of the puzzle. M can do this easily, since he holds the decryption key d. (Note that M cannot link this interaction with B as z' is randomized and cannot be related to z.) This involves an interactive fair-exchange protocol between A and M which allows A to get the puzzle's solution while allowing M to obtain a release transaction for the escrow they set up during the previous phase, signed only by her. Finally, A sends ε to B who computes $\varepsilon = \varepsilon'/r$ and checks whether $\varepsilon^e = z \bmod N$ (in which case he "accepts" A's payment). The fair exchange protocol guarantees that A gets the solution to the RSA puzzle if-and-only-if M gets a release on the escrowed transaction.

Lastly, during a cash-out phase, M signs his part of the release transaction for the escrow A set up and B uses ε to retrieve the encrypted signature by M on their escrow, which he additionally signs himself. Both parties post the signed release escrow transactions claiming the escrowed values and this concludes the protocol, since the bitcoins "traveled" from A to B via M.

Assuming k sender/recipient pairs during a single TumbleBit epoch, all of which mix the same value, the anonymity property achieved by TumbleBit guarantees that M cannot deduce the corresponding sender for a given recipient, based on his entire epoch view (expect with probability $1/k$). To avoid leaking additional information based on the timing of different protocol phases, all mixing transaction phases are synchronized and take a predetermined amount of time. Moreover, the fair-exchange protocol guarantees that as

c https://bitmixer.io
d https://bitlaunder.com
e https://helix-light.com

soon as M provides a solution to the puzzle, he receives the information he needs to claim A's escrow. Finally, the properties of the fair exchange protocol also guarantee that this will only happen if M provides the correct solution which implies B is able to claim the escrow set up by M (note that A is always motivated to send the solution to B as the bitcoins she escrowed will be claimed by M even if she does not).

Peer-to-peer mixing solutions. Next we turn our attention to alternative approaches that obviate the need for an intermediate party. One obvious benefit of this approach is that it eliminates the need for mixing fees. Moreover, it is closer in spirit to the decentralized principle behind Bitcoin; if the participants can themselves perform this service, why rely on a central provider?

Mixing with a single transaction. Each Bitcoin transaction can contain multiple input and output addresses. This allows a user to join inputs from multiple addresses she owns in order to match the cost of a particular goal. For example, if Alice is required to transfer 5BTC to Bob as part of a purchase, Alice can combine 2BTC from one address she owns and 3BTC from another, as inputs to a transaction that transfers 5BTC to an address owned by Bob. However, the Bitcoin protocol does not explicitly require that all input addresses belong to the same party. Multiple parties can, in principle, contribute input addresses to the same transaction (as shown in Figure 2). CoinJoin[20] is a mixing approach proposed by Maxwell that takes advantage of this liberty that Bitcoin offers. A set of k users can agree to jointly create a transaction with k input addresses that transfers its inputs to k output addresses. Each party individually observes the transaction; if her own output address appears in the list of recipients, she signs the transaction as a payer with her private key. Eventually, the transaction carries k different signatures. This simple idea has served as the core of multiple subsequent implementations and optimizations.

Internal Unlinkability. While CoinJoin hides the shuffling of the coins from an outsider (thus providing *external unlikability*), participants trivially learn the mapping from input to output addresses (that is, it lacks *internal unlikability*). CoinShuffle[33] avoids this by utilizing an anonymous group communication protocol that can hide the participants' identities from each other. This is achieved with the simple trick of layered encryption, as shown in Figure 3 (for four parties).

Assume three parties A, B, and C, with corresponding public keys pk_A, pk_B, pk_C, that want to mix the same amount of bitcoins each by transferring them to addresses A', B', and C', respectively. A then encrypts A', in a layered manner, first under pk_C and then under pk_B, that is, computes $Enc_{pk_B}(Enc_{pk_C}(A'))$. Likewise, B encrypts B' under pk_C to get $Enc_{pk_C}(B')$. Then, A sends the encryption of A' to B who proceeds to remove the outer encryption layer (using her own decryption key), randomly shuffles the resulting encryption with her own encryption of B', and forwards both to C. At this point C receives A', B' encrypted under pk_C and has no way of guessing which belongs to whom. She simply decrypts these values, appends C', shuffles all of them and writes the transaction which is broadcast to all participants. Each one checks that her recipient address is in the receivers list and, if so, signs the transaction. Once all signatures are gathered, the transaction is published

Figure 2. Example of decentralized mixing with four participants. Only the parties learn the mapping from input to output addresses.

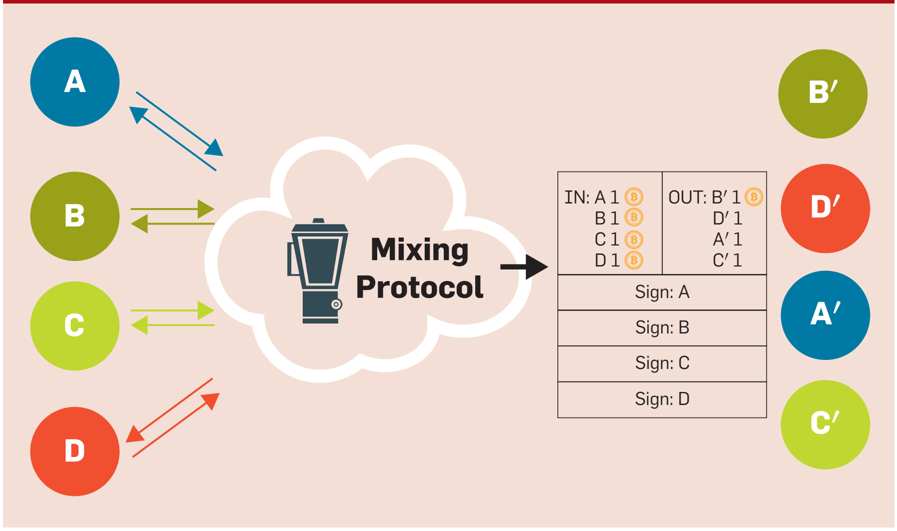
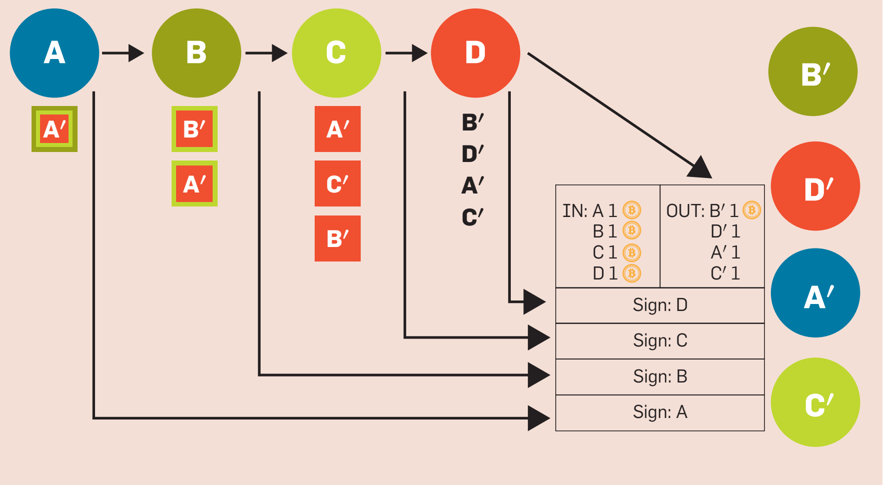


Figure 3. Decentralized mixing with CoinShuffle.

Each party encrypts her recipient address, in a layered manner, under the keys of all parties to her right and sends the shuffled vector of all addresses she sees to the next party who then removes the outermost layer (a colored rectangle denotes encryption with the key of the party with the corresponding color). The final party compiles the Bitcoin transaction and posts it on the blockchain. Everyone checks their recipient address is included and signs the transaction.



in the blockchain. CoinShuffle++[34] is an extension that uses a P2P network for traffic mixing while significantly reducing the performance and communication bandwidth.

Decentralized mixing with large anonymity sets. One issue with the peer-to-peer approaches is that their anonymity set is upper bounded by the number of participants in the mixing protocol, which is likely to be much smaller than that achieved by a "popular" centralized mixer (as we will discuss). One of the reasons is that typically the produced mixing transaction will have to carry a signature by each of the participants (for example, see figures 2 and 3). The total length of all these signatures blows up the size of the posted transaction significantly for larger sets, to the point that it may grow past the limits specified by Bitcoin (100KB for standard transactions). For example, Ruffing[34] is limited to 538 participants due to this.

In order to avoid this limitation, CoinParty[39] uses secure multiparty computation protocols that allow a set parties to collectively compute over their inputs in a way that does not reveal each party's input to other participating parties. Using such a protocol, the mixing participants collectively set up a single shared address (with off-chain communication) that is then used to transfer coins to fresh addresses. This means that the resulting transaction will only carry a single signature under this shared address. One major disadvantage of CoinParty is it requires at least 2/3 of the participants to be honest (which is an artifact of the secure multiparty protocol it uses), in order to guarantee no misbehavior with respect to the output signature.

Xim[6] can achieve large anonymity sets by an entirely different approach. Xim is a two-party mixing protocol that works as follows. First, during a pairing phase a party Alice that is interested in mixing her coins "advertises" this on the blockchain by posting a transaction that states she can be reached in a specific anonymous location (for example, a bulletin board maintained at a `.onion` Tor address she controls). An interested mixing partner Bob accesses the location expressing his interest by sending an anonymous location of his own (note that this communication takes place off the chain). After a specified amount of time, Alice chooses one of the interested partners that reached out to her (for example, Bob) and commits to proceeding by posting on her location a signed attestation of this. Within a fixed amount of time the two parties should post two transactions that "announce" their mutual pairing interest in a way that does not link their identities to outside observers. If that occurs, the two parties proceed to perform a single-transaction mixing, using a fair exchange protocol. If Bob backs down and does not post his transaction, Alice can simply announce she is looking for a new partner (without losing any funds) and if Alice backs down, Bob can post her signed attestation that confirms she changed her mind, "damaging" her reputation. Due to its interaction structure, Xim can achieve large anonymity sets, similar to the ones achieved by centralized mixers, assuming many participants are choosing to use it. The main downside is that it requires a significant blow-up in the end-to-end mixing time. A large portion of the communication happens sequentially over the chain itself therefore the waiting time for transactions to be collected by miners, added to blocks, posted to the chain, and substantially validated (by extending the chain) will typically be in the order of hours.

Alternative Privacy-Preserving Cryptocurrencies

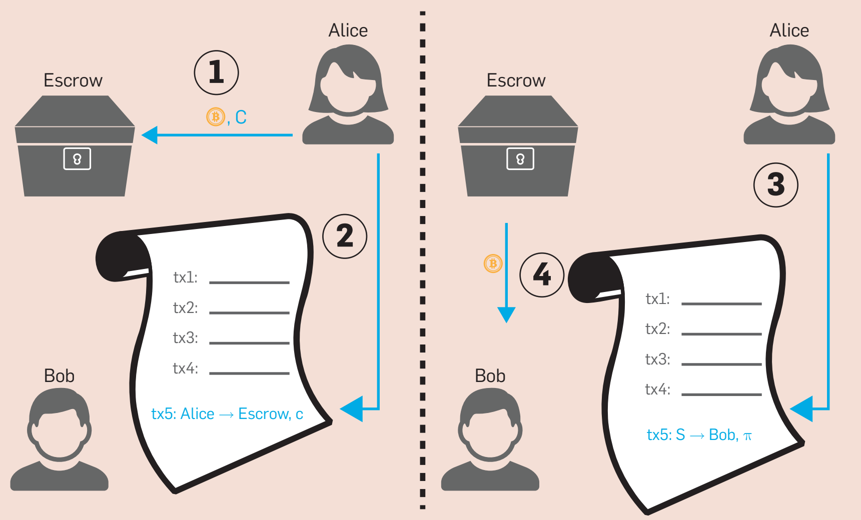
Here, we review some suggestions for alternative cryptocurrencies designed with the goal of providing stronger privacy guarantees than Bitcoin.

Privacy via ring signatures: CryptoNote. One of the first attempts to make transactions more private without additional interaction from the client (for example, using a mixing service or protocol) is CryptoNote,[38] the core idea of which has subsequently been refined and adopted in other currencies, for example, Bytecoin,[f] and Monero.[29] Like Bitcoin, every CryptoNote user has a public and a private key. Unlike Bitcoin however, the destination address of a transaction is a one-time public key, which is derived from the recipient's public key and some randomness chosen by the sender.

In particular, when Alice wants to send an amount m to Bob, she first establishes a one-time public key $pk_{B,r}$ with Bob using fresh randomness r and Bob's public key B. Then she posts a transaction on the blockchain that contains m, $pk_{B,r}$ and some pub-

Figure 4. Overview of Zerocoin.

(1–2) Alice places a coin with (hidden) serial number S and (visible) commitment c to escrow, by posting a corresponding transaction to the blockchain. (3–4) To pay Bob, Alice publishes a transaction with Bob as the receiver but no explicit sender. Instead of the sender, the transaction reveals S and a proof that it matches some coin in escrow. Everyone can check the validity of π but nobody can link the transaction to Alice.



f https://bytecoin.org

lic transaction information p. Next, Alice must sign the transaction in a way that proves her ownership of the funds being transferred but does not reveal her identity. Indeed, in order to do this Alice signs the transaction using the one-time secret key $sk_{A,r'}$ established during the transaction through which she originally acquired the funds. While this proves Alice's ownership over the transferred funds (since Alice is the only one that knows $sk_{A,r'}$, using regular digital signatures for the signature would trivially link the two transactions.

To prevent such leakage, CryptoNote uses ring signatures[12] to verify that a specific message was signed by some user belonging to a group of users, without revealing the signer's specific identity. Thus, Alice creates a transaction (which includes $pk_{B,r}$ as well as additional information p that will allow Bob to recover $sk_{B,r'}$ and choses some set of public keys PK which includes $pk_{A,r}$ from the public ledger. Alice then signs the transaction using $sk_{A,r'}$ and publishes the transaction and PK on a public ledger, thereby proving her ownership over the coin. Due to the hiding property of ring signatures, the signature may have originated from any of the users in PK. Thus, Alice is able to control the anonymity level of her transaction with Bob by simply varying the size of the set PK.

Zero-knowledge transactions: Zerocoin. As described earlier, the anonymity level provided by CryptoNote is directly related to the size of the set PK. However, the size of PK also affects the amount of work required by Alice in order to perform a transaction, thus hampering the performance of the cryptocurrency. Alleviating this issue, Zerocoin[25] uses a different approach that decouples the amount of work required by Alice from the achieved level of anonymity. Zerocoin works as an "overlay" over the Bitcoin protocol as follows: Assume Alice wishes to spend a (predetermined) number of bitcoins privately, without revealing her identity. The first step she takes is to *mint* a zerocoin by generating a random serial number S and by creating a commitment c to S using randomness r. Alice then publishes a transaction (for the amount she wishes to spend) from her address using c as destination (at this point, c can be seen as being held in escrow). The commitment c is then added by the network to a global, publicly visible set C of minted coins.

When Alice wishes to spend her new zerocoin, she creates a noninteractive zero-knowledge-proof-of-knowledge (NIZKPoK)[35,g] proof π of the statement "S is a valid opening to some commitment on an unspent zerocoin currently being held in escrow." Next, Alice publishes a transaction with Bob's address as destination and with an empty origin address containing π and S. At this point, due to the zero-knowledge property of π, there is no way to link Alice to any specific zerocoin commitment c.[h] The network accepts this transaction published by Alice only if the validation of π succeeds and S has not been previously spent. In this case, participants add S to the list of previously spent coins (see Figure 4).

Practical considerations. To minimize the size of the proof π, Zerocoin[25] implements the coin set C as an *accumulator*,[9] which is a cryptographic construction that allows efficient insertions and proofs of membership. Still, each spending transaction is 48KB (for 128-bit security level), exceeding the 10KB current limit for Bitcoin transactions. Also, note that the Bitcoin's source code does not support the necessary cryptographic operations.

Transactions with zk-SNARKs: Zerocash. Zerocash[2] is an alternative cryptocurrency that, unlike Zerocoin, hides both origin and destination addresses. Compared to Zerocoin, it provides additional functionality, that is, it handles transactions of arbitrary denominations, and it provides a way to give "change" after a transaction. Moreover, it improves Zerocoin's verification efficiency and proof size.

Protocol overview. Similar to Bitcoin, a Zerocash user Alice has a Zerocash address consisting of a public and secret key pair (pk_A, sk_A). Similar to Zerocoin, a coin c of value v is minted by having Alice sample a random serial number S and compute a commitment to the coin's value, serial number, and her public key pk_A. Next, Alice publishes a mint transaction that sends v bitcoins to the previously computed commitment c. As a result, the coin is being held in escrow and can only be spent by a user that knows Alice's secret key sk_A.

When Alice now wants to send v coins to Bob, she performs a *pour* transaction that is somewhat similar to the mint transaction: she posts a new transaction with a new coin c' with serial number S' but this time she ties c' it to Bob's public key pk_B; and she does not reveal her public address. Next, she computes a zero-knowledge succinct non-interactive argument of knowledge (zk-SNARK)[13] proof π to the following claim: "(1) S is a valid opening to some unspent coin c tied to an address pk_A currently held in escrow; (2) I know the secret key sk_A corresponding to pk_A; (3) c' has the same value v as c." Alice publishes a zerocash transaction containing S, π, c' without mentioning Bob's public address. The network accepts Alice's transaction only if π verifies and S has not been previously spent. In this case, participants add S to the list of previously spent coins. Notice that unlike Zerocoin, Bob's public address is not included as part of Alice's transaction. In fact, the only information that ever appears in the ledger in plaintext is the serial number of spent coins. Monitoring the ledger, Bob can test if a new coin c' was sent to him by testing it using his secret key sk_B. At that point, Bob can spend the coin as he wishes.

Implementing the set of committed coins. Zerocash does not use an accumulator[9] for the set of committed coins. Instead it uses Merkle hash trees[24] along with zk-SNARKs proofs.[13] Merkle trees have the same interface with RSA accumulators (they allow efficient insertion of elements and proofs of membership) but can be encoded in a zk-SNARK proof much more efficiently when a "SNARK-friendly" collision-resistant hash function is used.

Practical considerations. The use of zk-SNARKs drastically changes the performance of Zerocash from that of Zerocoin. In particular, the spending transaction size is reduced to under 1KB and its verification time is less than 6ms. On the other hand, creating this transaction takes significantly longer as the zk-SNARK prover algorithm is particularly demanding (however, this may be smaller than the block creation

g NIZKPoKs are cryptographic systems similar to zk-SNARKs but achieving weaker performance guarantees for the verifier.

h Recall that unlike a regular Bitcoin transaction, Alice did not publish her identity and the transaction's sender.

time). In October 2016, Zcash[i]—a cryptocurrency based on Zerocash—was officially launched. As of Jan. 26, 2017, Zcash has a market capitalization of \$20.5 million. It uses a mining mechanism similar to that of Bitcoin but based on an alternative memory-hard proof-of-work function[3] and it has four times smaller expected block creation time. The developers of Zcash chose to establish the public parameters upon which its security is bootstrapped via a secure multiparty computation protocol executed with a ceremony held among remote practitioners (some of which remained anonymous) and with several defense mechanisms deployed.[j]

Privacy beyond transactions: Hawk. The cryptocurrencies discussed so far aim to provide a single, basic functionality: transferring funds from Alice to Bob. However, imagine we had to implement a more complicated *contract* to decide how money would flow. For example, consider a Vickrey auction for some item offered by a seller S, where the transfer of money from Alice or Bob to S would depend on who made the highest bid. That person would finally take the item and pay the second highest price to S. Ethereum[k] is an alternative cryptocurrency aimed at securely executing such smart contracts on top of a blockchain-like public ledger. Unfortunately, Ethereum offers very weak privacy guarantees, revealing the sender's and receiver's addresses as well as all information and internal values computed inside the smart contract (in the example here, the bids of each user would be eventually leaked). Hawk[19] aims to offer notions of privacy while preserving arbitrary smart-contract functionality. The main protocol involves a party called the *manager* who is trusted for keeping participants' values (bids) secret, but not for executing the contract correctly.

At a high level, the protocol starts by having Alice and Bob *mint* a certain number of Hawk coins, say h_a and h_b, as in Zerocash and Zerocoin. Then, to participate in the auction there is a bidding period where Alice and Bob commit to their bids x_a and x_b using a hiding commitment, also computing a zk-SNARK proof they have minted enough coins to support their bids. When the bidding period ends, Alice and Bob post an encryption of their plaintext bids on the blockchain under the manager's public key, along with a zk-SNARK proof they have encrypted the same value as they committed in the bidding phase. Then the manager retrieves the plaintext values x_a and x_b, and redistributes the money based on the user-defined program that it executed privately, for example, an auction in this case, so the money goes from the winner to the seller, without leaking any information to the public. The manager submits a zk-SNARK proof indicating the correct execution of the private auction program, and the correct redistribution of money based on the private output of the program. Finally, the seller gets the new coins but nobody with access to the blockchain can find out who the winner was (assuming the manager does not leak the bids when running the auction).

In terms of concrete performance, assuming an auction with 100 participants, each one needs to publish two separate statements in the blockchain in preparation for the auction. The manager then publishes a final statement that concludes the auction. Each participant spends approximately 35sec preparing these statements in a phase that requires 4GB in memory. The corresponding costs for the manager are 3 minutes and 27GB.

Comparison of Existing Schemes

Next we attempt a comparison of the approaches discussed so far.

Mixing services. First, we compare mixing schemes in terms of their features (see the accompanying table, which is largely based on a similar comparison from Heilman[16]). We note that all of them are fully compatible with Bitcoin and do not require any modification in the codebase.

Decentralized protocols on the one hand avoid the need for a third party that in practice may become a single point of failure. However, they have the added issue of requiring participant coordination ahead of time in order to identify peers and form transactions. Also, the communication cost often scales quadratically in the number of participants, which in practice significantly limits the size of the anonymity set. For instance, none of CoinShuffle, CoinParty, or CoinShuffle++ scale the experimental evaluation they provide to more than 50 participants. Moreover, decentralized approaches are likely to achieve a quantitatively weaker privacy notion than the centralized solutions as, in contrast to the latter that hide an output address within the set of all mixer clients (input addresses) for a given time period, the

i https://z.cash
j https://goo.gl/fmHqUk
k https://www.ethereum.org

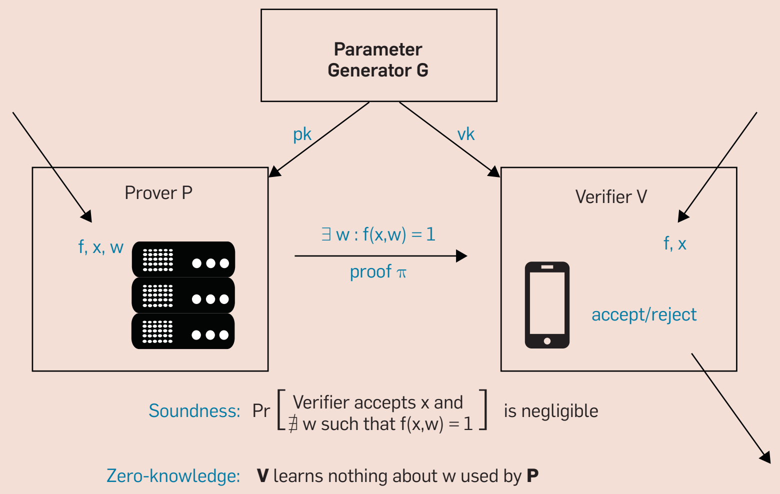
Comparison of the features of existing mixing schemes.

One bitcoin denotes a scheme fully achieves a property.
A parenthesis after an × in Unlinkable denotes which parties learn the link between input and output addresses.

	Avoids Coin-theft	Unlinkable	Anonymity Set	Adopted in practice
Untrusted mixer	×	× (mixer)	large	Bitmixer, Bitlaunder, Helix
Mixcoin[8]	accountable	× (mixer)	large	×
BlindCoin[37]	accountable	₿	large	×
CoinSwap[21]	₿	× (mixer)	large	×
TumbleBit[16]	₿ ₿	₿	large	Stratis[a]
CoinJoin[20]	₿	× (internal)	small	JoinMarket, DarkWallet, SharedCoin, Dash
CoinParty[39]	2/3 honest	₿	large	×
CoinShuffle[33]	₿	₿	small	Shufflepuff[b]
Coinshuffle++[34]	₿	₿	small	×
Xim[6]	₿	₿	large	×

a https://goo.gl/HXcr4J
b https://goo.gl/cCS2jz

The zk-SNARK Protocol


Parameter Generator G
pk
vk
Prover P
f, x, w
∃ w : f(x,w) = 1
proof π
Verifier V
f, x
accept/reject
Soundness: Pr [Verifier accepts x and ∄ w such that f(x,w) = 1] is negligible
Zero-knowledge: V learns nothing about w used by P

A *zero-knowledge succinct non-interactive argument of knowledge* (zk-SNARK)[13] is a protocol that allows a prover to prove claims of the form "I know *w* such that the output of program *P* on input *x*, *w* is 1" for pre-agreed program *P*. Crucially, the time it takes the verifier to check the validity of the prover's claim is much smaller than the time to run *P*(*x*,*w*). Moreover, during this verification process a polynomial-time verifier learns almost no information about *w* and the proof size generated by the zk-SNARK is short (for example, 288 bytes for Zerocash). While zk-SNARKs can be used to verify the execution of arbitrary programs, they have one notable downside. The public parameters used for proof construction and verification must be generated in a preprocessing phase by a trusted party. This raises the question of who can be entrusted to generate (and "forget") these parameters and opens a window of opportunity for an attacker to compromise the security of the system.

former hide it only within the set of participants of the particular transaction, which will typically be smaller (with the exception of Xim and Coinparty). The need to achieve larger anonymity sets (restricted only by the maximum transaction size and the hardness of coordinating) has given rise to services that "connect" interested users (for example, JoinMarket[l] for CoinJoin) operating as public bulletin boards and support for CoinJoin by existing wallets (for example, SharedCoin[m] and Darkwallet[n]). Note that the former was integrated to the popular blockchain.info wallet but support for it has since been suspended, partially due to issues related with limited privacy.[o] Finally, the idea behind CoinJoin has served as the core of for the alternative cryptocurrency Dash[p] that achieves large anonymity sets.

Sybil attacks (where an attacker poses as multiple mixing users in order to reduce the size of the anonymity set of honest participants) are a common problem for the above proposals. One partial countermeasure is imposing a "participation fee" that is payable by every user that wishes to mix her coins.[6,8,16,21,37,39] Finally, one technique that can be applied on top of some of these schemes (for example, Ruffing[32]) in order to hide the amount exchanged in the transaction is Confidential Transactions.[q]

Alternative cryptocurrencies. Among the cryptocurrencies we reviewed in this article, there exist two notable trade-offs. The privacy provided by CryptoNote to the transaction sender (Alice) directly depends on the size of the group Alice choses to participate in her ring signature. Moreover, Alice must publish the public keys of all the chosen group members. Thus, in order to remain *completely* undetectable Alice must use the public keys of all the users for her ring signature, making signature creation and verification expensive. On the other hand, Zerocoin and Zerocash achieve by default the "maximal" anonymity set, as all transactions ever to be published seem identical. However, they come with a drawback of their own. Zerocoin and Zerocash require a trusted party in order to setup the public parameters (for example, the RSA modulo for the former and the zk-SNARK parameters for the latter). While this only takes place once, as discussed above, any successful attack on the trusted party (including the party itself misbehaving) results in a complete compromise of the coins' security. Finally, Zerocash is much more efficient than Zerocoin, however it relies on much stronger "nonfalsifiable" cryptographic assumptions.[14]

Overall comparison. Attempting to compare these two "classes" of privacy techniques, one major drawback of mixing-based privacy solutions is they require various degrees of interaction from the client (either with the mixer or with other clients) in order to ensure privacy, which may impair their practical adoption. However, these solutions run on top of the widely used Bitcoin. On the other hand, any alternative cryptocurrency requires a significant amount of time for the community to become familiar with as well as to test it and trust it. As most of these protocols require a large crowd-base size in order to achieve strong security properties, this becomes an inhibiting factor for every new proposal. The main advantage of privacy-preserving cryptocurrencies is they increase the client's anonymity set from relatively small sets of clients that use a particular mixing service or participate in a transaction, to large sets that include *all* the users of a given cryptocurrency.

Discussion

We believe our exposition so far indicates there is no general consensus regarding a technique for anonymous cryptocurrencies. This should come as no surprise given the relative infancy of the field and the fact that different participants may have different privacy requirements. For example, for most users it may be sufficient to run a single round of CoinJoin with a dozen users whereas privacy-aware users

l https://github.com/JoinMarket-Org/joinmarket
m https://en.bitcoin.it/wiki/Shared_coin
n https://www.darkwallet.is
o http://www.coinjoinsudoku.com
p https://www.dash.org

q https://bitcointalk.org/index.php?topic=1085273 may choose to opt for something more thorough. Moreover, there exist other approaches for privacy that do not fall within any of the two categories, for example, private payments in credit networks[26] and payment channels.[15,r] Next, we discuss a number of open problems that arise while trying to design better private cryptocurrencies.

Unified formal privacy definition. One particular issue has to do with the formal treatment of the problem. While some existing works attempt to provide a definition of anonymity in the context of cryptocurrencies (for example, Bonneau[8] and Meiklejohn[22] for mixers and Ben-Sasson[2] and Miers[25] for alternative cryptocurrencies), there is no de facto unified privacy definition that would allow a fair comparison of different proposals (for example, it is difficult to quantitatively compare the security properties of Zerocash and Cryptonote if they satisfy different privacy definitions). Due to the nature and scale of cryptocurrency implementations, one very robust (but challenging in formulation) framework would be that of universal composability,[10] along the lines of the one introduced in Kosba[19] for private smart contracts.

Strong anonymity with milder sssumptions. A more concrete problem has to do with designing cryptocurrencies that achieve the strong anonymity levels of Zerocash but without the need for a sensitive trusted setup phase and without relying on the non-falsifiable cryptographic assumptions inherent to zk-SNARKs. The problem becomes even more important in the context of smart contracts as Hawk requires a separate trusted setup process for the generation of each different contract.

Scalable anonymous cryptocurrencies. Perhaps the most important challenge for Bitcoin (and other cryptocurrencies) is scalability; for any privacy solution to be widely used in practice, it must not only protect the users' anonymity but also be able to scale to realistic numbers of users and transactions. For example, Zerocash[2] reports more than 40 seconds of proving time per transaction and requires approximately 1GB of memory. Both of these inhibit the potential of large-scale deployments.

Privacy abuse and stricter policies. While the goal of this article has been to provide an overview of techniques for achieving anonymity in cryptocurrencies, it should be noted that increased user privacy may raise concerns, such as users participating in illegal activities[18] or facilitating various cryptographic ransomware.[s] This in turn may lead to stricter government regulation of cryptocurrency transactions[t] and requests for auditability,[u] which seems inherently incompatible with the need for stronger user anonymity.

Acknowledgments

This work was supported in part by NSF awards #1514261 and #1652259, financial assistance award 70NANB15H328 from the U.S. Department of Commerce, National Institute of Standards and Technology, the 2017-2018 Rothschild Postdoctoral Fellowship, and the Defense Advanced Research Project Agency (DARPA) under Contract #FA8650-16-C-7622. C

r For detailed presentation, see https://z.cash/static/R3_Confidentiality_and_Privacy_Report.pdf

s https://goo.gl/8Eujkr
t https://goo.gl/4WueYR
u https://goo.gl/iUnNMQ

References

1. Androulaki, E., Karame, G., Roeschlin, M., Scherer, T. and Capkun, S. Evaluating user privacy in Bitcoin. In *Proceedings of FC 2013*, 34–51.
2. Ben-Sasson, E., Chiesa, A., Garman, C., Green, M., Miers, I., Tromer, E. and Virza, M. Zerocash: Decentralized anonymous payments from Bitcoin. In *Proceedings of IEEE SP 2014*, 459–474.
3. Biryukov, A. and Khovratovich, D. Equihash: Asymmetric proof-of-work based on the Generalized Birthday Problem. In *Proceedings of NDSS 2016*.
4. Biryukov, A. and Khovratovich, D. and Pustogarov, I. Deanonymisation of clients in Bitcoin P2P network. In *Proceedings of ACM CCS 2014*, 15–29.
5. Biryukov, A. and Pustogarov, I. Bitcoin over Tor isn't a Good Idea. In *Proceedings of IEEE SP*, 2015, 122–134.
6. Bissias, G.D., Ozisik, A.P., Levine, B.N. and Liberatore, M. Sybil-resistant mixing for Bitcoin. In *Proceedings of WPES*, 2014, 149–158.
7. Bonneau, J., Miller, A., Clark, J., Narayanan, A., Kroll, J.A. and Felten, E.W. SoK: Research perspectives and challenges for Bitcoin and cryptocurrencies. In *Proceedings of IEEE SP*, 2015, 104–121.
8. Bonneau, J., Narayanan, A., Miller, A., Clark, J., Kroll, J.A. and Felten, E.W. Mixcoin: Anonymity for Bitcoin with accountable mixes. In *Proceedings of FC*, 2014, 486–504.
9. Camenisch, J. and Lysyanskaya, A. Dynamic accumulators and application to efficient revocation of anonymous credentials. In *Proceedings of CRYPTO*, 2002, 61–76.
10. Canetti, R. Universally composable security: A new paradigm for cryptographic protocols. In *Proceedings of FOCS*, 2001, 136–145.
11. Chaum, D. Blind signatures for untraceable payments. In *Proceedings of CRYPTO '82*, 199–203.
12. Fujisaki, E. and Suzuki, K. Traceable ring signature. In *Proceedings of PKC*, 2007, 181–200.
13. Gennaro, R., Gentry, C., Parno, B. and Raykova, M. Quadratic span programs and succinct NIZKs without PCPs. In *Proceedings of EUROCRYPT*, 2013, 626–645.
14. Gentry, C. and Wichs, D. Separating succinct non-interactive arguments from all falsifiable assumptions. In *Proceedings of STOC*, 2011, 99–108.
15. Green, M.D. and Miers, I. Bolt: Anonymous payment channels for decentralized currencies. *IACR Cryptology ePrint Archive*, 2016, 701.
16. Heilman, E., Baldimtsi, F., Alshenibr, L., Scafuro, A. and Goldberg, S. TumbleBit: An untrusted tumbler for Bitcoin-compatible anonymous payments. In *Proceedings of NDSS*, 2017.
17. Hileman, G. and Rauchs, M. Global cryptocurrency benchmarking study. *Cambridge Centre for Alternative Finance Global Cryptocurrency Benchmarking Study*, 2017.
18. Juels A., Kosba, A. E., and Shi, E. The ring of Gyges: Investigating the future of criminal smart contracts. In *Proceedings of ACM CCS*, 2016, 283–295.
19. Kosba, A.E., Miller, A., Shi, E, Wen, Z., and Papamanthou, C. Hawk: The blockchain model of cryptography and privacy-preserving smart contracts. In *Proceedings of IEEE SP*, 2016, 839–858.
20. Maxwell, G. CoinJoin: Bitcoin privacy for the real world. bitcointalk.org, Aug. 2013.
21. Maxwell, G. CoinSwap: Transaction graph disjoint trustless trading. bitcointalk.org, Oct. 2013.
22. Meiklejohn, S. and Orlandi, C. Privacy-enhancing overlays in Bitcoin. In *Proceedings of FC Workshops, BITCOIN, WAHC, and Wearable*, 2015, 127–141.
23. Meiklejohn, S., Pomarole, M., Jordan, G., Levchenko, K., McCoy, D., Voelker, G. M., and Savage, S. A fistful of Bitcoins: Characterizing payments among men with no names. In *Proceedings of IMC*, 2013, 127–140.
24. Merkle, R.C. A certified digital signature. In *Proceedings of CRYPTO '89*, 218–238.
25. Miers, I., Garman, C., Green, M., and Rubin, A.D. Zerocoin: Anonymous distributed e-cash from Bitcoin. In *Proceedings of IEEE SP*, 2013, 397–411.
26. Moreno-Sanchez, P., Kate, A., Maffei, M., and Pecina, K. Privacy preserving payments in credit networks: Enabling trust with privacy in online marketplaces. In *Proceedings of NDSS*, 2015.
27. Möser, M. An Inquiry into Money Laundering Tools in the Bitcoin Ecosystem. In *IEEE 2013 eCrime Researchers Summit*.
28. Nakamoto, S. Bitcoin: A Peer-to-Peer Electronic Cash System, 2008; http://bitcoin.org/bitcoin.pdf.
29. Noether, S., Mackenzie, A., and the Monero Research Lab. Ring confidential transactions. *Ledger 1* (2016) 1–18.
30. Reid, F. and Harrigan, M. An analysis of anonymity in the Bitcoin system. In *Proceedings of IEEE PASSAT and SocialCom*, 2011, 1318–1326.
31. Ron, D. and Shamir, A. Quantitative analysis of the full Bitcoin transaction graph. In *Proceedings of FC 2013*, 6–24.
32. Ruffing, T. and Moreno-Sanchez, P. Mixing confidential transactions: Comprehensive transaction privacy for bitcoin. *IACR Cryptology ePrint Archive*, 2017, 238.
33. Ruffing, T., Moreno-Sanchez, P., and Kate, A. CoinShuffle: Practical decentralized coin mixing for Bitcoin. In *Proceedings of ESORICS*, 2014.
34. Ruffing, T., Moreno-Sanchez, P., and Kate, A. P2P mixing and unlinkable Bitcoin transactions. In *Proceedings of NDSS*, 2017.
35. Sahai, A. Non-malleable non-interactive zero knowledge and adaptive chosen ciphertext security. In *Proceedings of FOCS '99*, 543–553.
36. Sweeney, L. k-Anonymity: A model for protecting privacy. *Intern. J. Uncertainty, Fuzziness and Knowledge-Based Systems 10*, 5 (2002), 557–570.
37. Valenta, L. and Rowan, B. Blindcoin: Blinded, accountable mixes for Bitcoin. In *Proceedings of the 2015 FC International Workshops, BITCOIN, WAHC, and Wearable*, 112–126.
38. van Saberhagen, N. CryptoNote v 2.0; https://cryptonote.org/whitepaper.pdf.
39. Ziegeldorf, J. H., Grossmann, F., Henze, M., Inden, N., and Wehrle, K. CoinParty: Secure multi-party mixing of Bitcoins. *CODASPY* (2015), 75–86.

Daniel Genkin (danielg3@cis.upenn.edu) is a postdoctoral researcher at the University of Pennsylvania, Philadelphia, and the University of Maryland, College Park, MD, USA.

Dimitrios Papadopoulos (dipapado@cse.ust.hk) is an assistant professor of computer science and engineering at Hong Kong University of Science and Technology.

Charalampos Papamanthou (cpap@umd.edu) is an assistant professor of electrical and computer engineering at the University of Maryland, College Park, MD, USA.

P. 90

Technical Perspective

Measuring Optimization Potential with Coz

By Landon P. Cox

P. 91

Coz: Finding Code that Counts with Causal Profiling

By Charlie Curtsinger and Emery D. Berger

DOI:10.1145/3205913

To view the accompanying paper, visit doi.acm.org/10.1145/3205911 rh

Technical Perspective
Measuring Optimization Potential with Coz

By Landon P. Cox

WHEN PROGRAMMERS WANT to improve their program's performance, they often turn to profilers to tell them what code to optimize. A profiler can measure many aspects of a program's behavior, such as how many times each method is typically called, how long each method typically takes, and which methods typically lie on the critical path. Unfortunately, this information is not always relevant, and it can often cause programmers to waste their time on optimizations that have little impact on overall performance.

In particular, conventional profilers struggle to help developers optimize multithreaded programs. A simple but useful example is a program in which an initial thread waits for several worker threads to complete. If each worker runs for approximately the same amount of time, then most profilers will report that each worker accounted for an equal share of the execution time. At the same time, optimizing any individual worker will have a minimal impact on overall execution time since the program will only finish after all workers are done. A programmer could waste countless hours optimizing code without making the program run any faster.

In the following paper, Curtsinger and Berger describe a better approach called causal profiling; causal profilers tell programmers exactly how much speed-up bang to expect for their optimization buck. That is, a causal profiler can predict with spooky accuracy that speeding up a line of code by *x*% will improve overall responsiveness or throughput by *y*%. At first blush this seems like magic. No tool can make an arbitrary line of code arbitrarily faster and then measure the sped-up line's impact on overall execution time. And yet Curtsinger and Berger developed a tool called Coz that would seem to do the impossible.

The key observations underlying Coz are that arbitrarily accelerating code is infeasible, but arbitrarily slowing code is quite easy, and all a tool has to do is adjust the relative speeds of code fragments in order to predict how a change in one fragment will impact overall performance. Put another way, Curtsinger and Berger realized one can measure the impact of speeding up a target code fragment by *x*% by slowing everything but the target by *x*% and measuring the overall slowdown. This wonderful technique is called a virtual speedup, and it is one of my favorite pieces of work in the past few years. Using virtual speedups for code profiling is such a clever, simple, and powerful idea that you wish you had thought of it yourself.

While the insights underlying Coz are undeniably clever, the implementation and evaluation results are equally impressive.

While the insights underlying Coz are undeniably clever, the implementation and evaluation results are equally impressive. Conceptually, whenever Coz runs a target line of code, it pauses all parallel threads by a delay that is less than the average runtime of the target line. However, naïvely pausing threads each time a target line executes would require heavy code instrumentation that could significantly skew the results and undermine the utility of the profiler. Instead, Coz uses a lightweight sampling scheme that requires no instrumentation at all. In Coz, each thread maintains a local counter and periodically samples its program counter and stack using Linux's interface to the CPU's hardware performance counters. When a thread finds it is executing the target method, it increments the global counter. When another thread processes its sample, it compares the value of its local counter to the global counter, and if the local counter is behind, increments the counter and pauses the thread. This sampling scheme is extremely elegant, and it allows Coz to set the virtual speedup of an execution by simply adjusting the ratio of the delay and the sampling period.

Finally, Coz's predictions proved to be accurate across a wide range of benchmarks. Maybe most impressive, the authors used Coz to fix a really nasty and longstanding performance bug in a deployed hash table. Coz reported the greatest optimization opportunity in a file-compression application was a method that traversed the linked list of a hash-table bucket. Coz continued to identify this particular line even after the authors increased the number of buckets. Upon closer inspection, the authors discovered the table's hashing function was assigning entries to only 3% of the available buckets. Fixing the buggy hash function required changing three lines of code and led to a nearly 9% end-to-end benchmark speedup. Coz not only identified the bug and predicted the impact of a fix, but it allowed the authors to discover and resolve the problem in just a few hours.

Above all else, what makes Coz noteworthy is that it exemplifies the best kind of systems work: it is elegant, insightful, and practical all at once. Papers that simultaneously achieve all of these qualities are exceedingly rare, and I suspect that practitioners and researchers will continue returning to it for many years to come. C

Landon P. Cox is an associate professor in the Department of Computer Science at Duke University, Durham, NC, USA.

DOI:10.1145/3205911

Coz: Finding Code that Counts with Causal Profiling

By Charlie Curtsinger and Emery D. Berger

Abstract

Improving performance is a central concern for software developers. To locate optimization opportunities, developers rely on software profilers. However, these profilers only report where programs spend their time: optimizing that code may have no impact on performance. Past profilers thus both waste developer time and make it difficult for them to uncover significant optimization opportunities.

This paper introduces *causal profiling*. Unlike past profiling approaches, causal profiling indicates exactly where programmers should focus their optimization efforts, and quantifies their potential impact. Causal profiling works by running *performance experiments* during program execution. Each experiment calculates the impact of any potential optimization by *virtually speeding* up code: inserting pauses that slow down all other code running concurrently. The key insight is that this slowdown has the same *relative* effect as running that line faster, thus "virtually" speeding it up.

We present Coz, a causal profiler, which we evaluate on a range of highly-tuned applications such as Memcached, SQLite, and the PARSEC benchmark suite. Coz identifies previously unknown optimization opportunities that are both significant and targeted. Guided by Coz, we improve the performance of Memcached by 9%, SQLite by 25%, and accelerate six PARSEC applications by as much as 68%; in most cases, these optimizations involve modifying under 10 lines of code.

1. INTRODUCTION

Improving performance is a central concern for software developers. While compiler optimizations are of some assistance, they often do not have enough of an impact on performance to meet programmers' demands.[2] Programmers seeking to increase the throughput or responsiveness of their applications thus must resort to manual performance tuning.

Manually inspecting a program to find optimization opportunities is impractical, so developers use profilers. Conventional profilers rank code by its contribution to total execution time. Prominent examples include oprofile, perf, and gprof.[7,9,11] Unfortunately, even when a profiler accurately reports where a program spends its time, this information can lead programmers astray. Code that runs for a long time is not necessarily a good choice for optimization. For example, optimizing code that draws a loading animation during a file download will not make the program run faster, even though this code runs just as long as the download.

This work was initiated and partially conducted while Charlie Curtsinger was a Ph.D. student at the University of Massachusetts Amherst.

This phenomenon is not limited to I/O operations. Figure 1 shows a simple program that illustrates the shortcomings of existing profilers, along with its gprof profile as shown in Figure 2a. This program spawns two threads, which invoke functions f_a and f_b, respectively. Most profilers will report that these functions comprise roughly half of the total execution time. Other profilers may report that f_a is on the critical path, or that the main thread spends roughly equal time waiting for f_a and f_b. While accurate, all of this information is potentially misleading. Optimizing f_a away entirely will only speed up the program by 4.5% because f_b becomes the new critical path.

Conventional profilers do not report the potential impact of optimizations; developers are left to make these predictions based on their understanding of the program. While these predictions may be easy for programs as simple as the one shown in Figure 1, accurately predicting the effect of a proposed optimization is nearly impossible for programmers attempting to optimize large applications.

This paper introduces *causal profiling*, an approach that accurately and precisely indicates where programmers should focus their optimization efforts, and quantifies their potential impact. Figure 2b shows the results of running Coz, our prototype causal profiler. This profile plots the hypothetical speedup of a line of code (x-axis) versus its impact on execution time (y-axis). The graph correctly shows that optimizing either f_a or f_b in isolation would have little effect.

A causal profiler conducts a series of *performance experiments* to empirically observe the effect of a potential optimization. Of course it is not possible to automatically speed up any line of code by an arbitrary amount. Instead, a causal profiler uses the novel technique of *virtual speedups* to

Figure 1. A simple multithreaded program that illustrates the shortcomings of existing profilers. Optimizing f_a will improve performance by no more than 4.5%, while optimizing f_b would have no effect on performance.

example.cpp

```
void a() { // ~6.7 seconds
  for (volatile size_t x=0; x<2000000000; x++) {}
}
void b() { // ~6.4 seconds
  for (volatile size_t y=0; y<1900000000; y++) {}
}
int main() {
  // Spawn both threads and wait for them.
  thread a_thread(a), b_thread(b);
  a_thread. join(); b_thread. join();
}
```

The original version of this paper was published in *Proceedings the ACM 2015 Symposium on Operating Systems Principles*, 184–197.

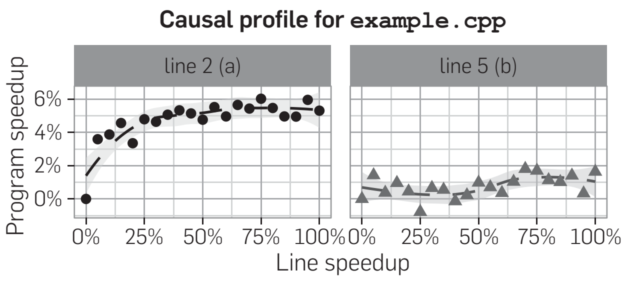
Figure 2. The gprof and causal profiles for the code in Figure 1. In the causal profile, the y-axis shows the program speedup that would be achieved by speeding up each line of code by the percentage on the x-axis. The gray area shows standard error. Gprof reports that f_a and f_b comprise similar fractions of total runtime, but optimizing f_a will improve performance by at most 4.5%, and optimizing f_b would have no effect on performance. The causal profile predicts both outcomes within 0.5%.

Conventional profile for `example.cpp`

```
    % cumulative      self              self    total
 time    seconds    seconds    calls  Ts/call  Ts/call     name
55.20       7.20       7.20        1                       a()
45.19      13.09       5.89        1                       b()

% time  self   children    called     name
  55.0  7.20      0.00                a()
-----------------------------------------------
  45.0  5.89      0.00                b()
```

(a) A gprof profile for `example.cpp`



(b) Causal profile for `example.cpp`

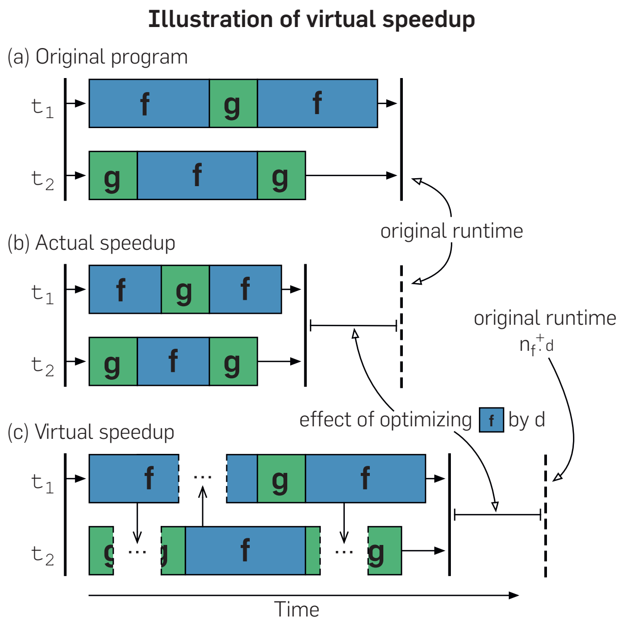
mimic the effect of optimizing a specific fragment of code by a fixed amount. Fragments could be functions, basic blocks, source lines. A fragment is virtually sped up by inserting pauses to slow all other threads each time the fragment runs. The key insight is that this slowdown has the same relative effect as running that fragment faster, thus "virtually" speeding it up. Figure 3 shows the equivalence of virtual and actual speedups.

Each performance experiment measures the effect of virtually speeding up a fragment of code by a specific amount. Speedups are measured in percent change; a speedup of 0% means the fragment's runtime is unchanged, while 75% means the fragment takes a quarter of its original runtime. By conducting many performance experiments over a range of virtual speedups, a causal profiler can predict the effect of any potential optimization on a program's performance.

Causal profiling further departs from conventional profiling by making it possible to view the effect of optimizations on both *throughput* and *latency*. To profile throughput, developers specify a *progress point*, indicating a line in the code that corresponds to the end of a unit of work. For example, a progress point could be the point at which a transaction concludes, when a web page finishes rendering, or when a query completes. A causal profiler then measures the rate of visits to each progress point to determine any potential optimization's effect on throughput. To profile latency, programmers instead place progress points at the start and end of an event of interest, such as when a transaction begins and completes. A causal profiler then reports the effect of potential optimizations on the average latency between those two progress points.

To demonstrate the effectiveness of causal profiling, we have developed Coz, a causal profiler for Linux. We show that causal profiling accurately predicts optimization opportunities, and that it is effective at guiding optimization efforts. We apply Coz to Memcached, SQLite, and the extensively studied PARSEC benchmark suite. Guided by Coz's output, we improve the performance of Memcached by 9%, SQLite by 25%, and six PARSEC applications by as much as 68%. Our changes typically require modifying under 10 lines of code. We also show that Coz imposes low execution overhead. When it is possible, we compare the observed performance improvements to Coz's predictions. In each case, we find that the real effect of our optimization matches Coz's prediction.

Figure 3. An illustration of virtual speedup: (a) shows the original execution of two threads running functions `f` and `g`; (b) shows the effect of a *actually* speeding up `f` by 40%; (c) shows the effect of *virtually* speeding up `f` by 40%. Each time `f` runs in one thread, all other threads pause for 40% of `f`'s original execution time (shown as ellipsis). The difference between the runtime in (c) and the original runtime plus $n_f \cdot d$—the number of times `f` ran times the delay size—is the same as the effect of actually optimizing `f`.



1.1. Contributions

This paper makes the following contributions:

1. It presents *causal profiling*, which identifies code where optimizations will have the largest impact. Using *virtual speedups* and *progress points*, causal profiling directly measures the effect of potential optimizations on both throughput and latency (Section 2).
2. It presents *Coz*, a causal profiler that works on unmodified Linux binaries. It describes Coz's implementation (Section 3), and demonstrates its efficiency and effec-

tiveness at identifying optimization opportunities (Section 4).

2. CAUSAL PROFILING OVERVIEW

This section describes the major steps in collecting, processing, and interpreting a causal profile with Coz, our prototype causal profiler.

Profiler startup. A user invokes Coz using a command of the form `coz run --- <program> <args>`. At the beginning of the program's execution, Coz collects debug information for the executable and all loaded libraries to build a source map. By default, Coz will consider speedups in any source file from the main executable. This means Coz will only test potential optimizations in the main program's source files. Users should use binary and source file scope options to specify exactly which code they are willing or able to change to improve their program's performance. Once the source map is constructed, Coz creates a profiler thread and resumes normal execution.

Experiment initialization. Coz's profiler thread begins an experiment by selecting a line to virtually speed up, and a randomly-chosen percent speedup. Both parameters must be selected randomly; any systematic method of exploring lines or speedups could lead to systematic bias in profile results. One might assume that Coz could exclude lines or virtual speedup amounts that have not shown a performance effect early in previous experiments, but prioritizing experiments based on past results would prevent Coz from identifying an important line if its performance only matters after some warmup period. Once a line and speedup have been selected, the profiler thread records the number of visits to each progress point and begins the experiment.

Applying a virtual speedup. Every time the profiled program creates a thread, Coz begins sampling the instruction pointer from this thread. Coz processes samples within each thread to implement a sampling version of virtual speedups. In Section 3.4, we show the equivalence between the virtual speedup mechanism as shown in Figure 3, and the sampling approach used by Coz. Each thread periodically processes its own samples; threads check whether any samples fall within the line of code selected for virtual speedup. If so, all other threads must pause. This process continues until the profiler thread indicates that the experiment has completed.

Ending an experiment. Coz ends the experiment after a predetermined time has elapsed. If there were too few visits to progress points during the experiment—five is the default minimum—Coz doubles the experiment time for the rest of the execution. Once the experiment has completed, the profiler thread logs the results of the experiment, including the effective duration of the experiment (runtime minus the total inserted delay), the selected line and speedup, and the number of visits to all progress points. Before beginning the next experiment, Coz will pause for a brief cooloff period to allow any remaining samples to be processed before the next experiment begins.

Producing a causal profile. After an application has been profiled with Coz, the results of all the performance experiments can be combined to produce a causal profile. Each experiment has two independent variables: the line chosen for virtual speedup and the amount of virtual speedup. Coz records the dependent variable, the rate of visits to each progress point, in two numbers: the total number of visits to each progress point and the effective duration of the experiment (the real runtime minus the total length of all pauses). Experiments with the same independent variables can be combined by adding the progress point visits and experiment durations.

Once experiments have been combined, Coz groups experiments by the line that was virtually sped up. Any lines that do not have a measurement of 0% virtual speedup are discarded; without this baseline measurement, we cannot compute a percent speedup relative to the original program. Measuring this baseline separately for each line guarantees that any line-dependent overhead from virtual speedups, such as the additional cross-thread communication required to insert delays when a frequently-executed line runs, will not skew profile results. By default, Coz also discards any lines with fewer than five different virtual speedup amounts (a plot that only shows the effect of a 75% virtual speedup is not particularly useful). Finally, we compute the percent program speedup for each grouped experiment as the percent change in rate of visits to each progress point over the baseline (virtual speedup of 0%). Coz then plots the resulting table of line and program speedups for each line, producing the profile graphs shown in this paper.

Interpreting a causal profile. Once causal profile graphs have been generated, it is up to the user to interpret them and make an educated choice about which lines may be possible to optimize. To help the user identify important lines, Coz sorts the graphs by the slope of their linear regression. Steep upward slopes indicate a line where optimizations will generally have a positive impact, while a flat line indicates that optimizing this line will not improve program performance. Coz also finds lines with a steep *downward* slope, meaning any optimization to this line will actually hurt performance. This downward sloping profile is a strong indicator of contention; the line that was virtually sped up interferes with the program's critical path, and optimizing this line increases the amount of interference. This phenomenon is surprisingly common, and can often result in significant optimization opportunities. In three of the eight applications we sped up using Coz—fluidanimate, streamcluster, and Memcached—the causal profile indicated contention issues. Fixing these issues resulted in speedups of 37.5%, 68.4%, and 9.4% respectively.

3. IMPLEMENTATION

This section describes Coz's basic functionality and implementation. We briefly discuss the core mechanisms required to support profiling unmodified Linux x86-64 executables, along with implementation details for each of the key components of a causal profiler: performance experiments, progress points, and virtual speedups.

3.1. Core mechanisms

Coz uses sampling to implement both virtual speedups and progress points. When a user starts a program with the `coz` command, Coz injects a profiling runtime library into the program's address space using `LD_PRELOAD`. This runtime library creates a dedicated profiler thread to run performance experiments, but also intercepts each thread startup and shutdown to start and stop sampling in the thread using the `perf_event` API. COZ collects both the current program counter and user-space call stack from each thread every 1ms. To reduce overhead, COZ processes samples in batches of ten by default (every 10ms). Batching samples increases the time between when a thread runs a virtually sped-up function and delaying other threads, potentially introducing some inaccuracy in virtual speedup; however, processing samples less frequently reduces the time spent running COZ instead of the program being profiled. Processing samples more frequently is unlikely to improve accuracy, as the additional overhead would distort program execution.

Attributing samples to source locations. COZ uses DWARF debug information to map sampled program counter values to source locations. The profiled program does not need to contain DWARF line information; COZ will use the same search procedure as GNU Debugger (GDB) to locate external debug information if necessary.[5] Note that debug information is available even for optimized code, and most Linux distributions offer packages that include debug information.

3.2. Performance experiments

COZ uses a dedicated profiler thread to coordinate performance experiments. This thread is responsible for selecting a line to virtually speed up, selecting the size of the virtual speedup, measuring the effect of the virtual speedup on progress points, and writing profiler output.

Starting a performance experiment. A single profiler thread, created during program initialization, coordinates performance experiments. First, the profiler selects a source line to virtually speed up. To do this, all program threads sample their instruction pointers and map these addresses to source lines. The first thread to sample a source line that falls within the specified profiling scope sets this as the line selected for virtual speedup.

Once the profiler receives a valid line from one of the program's threads, it chooses a random virtual speedup between 0% and 100%, in multiples of 5%. For any given virtual speedup, the effect on program performance is $1-\frac{p_s}{p_0}$, where p_0 is the period between progress point visits with no virtual speedup, and p_s is the same period measured with some virtual speedup s. Because p_0 is required to compute program speedup for every p_s, a virtual speedup of 0 is selected with 50% probability. The remaining 50% is distributed evenly over the other choices.

Lines for virtual speedup must be selected randomly to prevent bias in the results of performance experiments. A naive approach would be to begin conducting performance experiments with small virtual speedups, gradually increasing the speedup until it no longer has an effect on program performance. Using this policy, a line that has no performance impact during a program's initialization would not be measured later in execution when optimizing it could have significant performance benefit. Any systematic approach to exploring the space of virtual speedup values could potentially lead to systematic bias in the profile output.

Once a line and speedup amount have been selected, COZ saves the current values of all progress point counters and begins the performance experiment.

Running a performance experiment. Once a performance experiment has started, each of the program's threads processes samples and inserts delays to perform virtual speedups. After the predetermined experiment time has elapsed, the profiler thread logs the end of the experiment, including the current time, the number and size of delays inserted for virtual speedup, the running count of samples in the selected line, and the values for all progress point counters. After a performance experiment has finished, COZ waits until all samples collected during the current experiment have been processed. By default, COZ processes samples in groups of ten, so this pause time is just ten times the sampling rate of 1ms. Lengthening this cooloff period will reduce COZ's overhead by inserting fewer delays at the cost of increased profiling time to conduct the same number of performance experiments.

3.3. Progress points

Before a program can be profiled with COZ, developers must add at least one progress point to the program. To indicate a progress point, a developer simply inserts the `COZ_PROGRESS` macro in the program's source code at the appropriate location. COZ also has experimental support for progress points that use sampling and breakpoints, which we describe in our Symposium on Operating Systems Principles (SOSP) paper.[3]

Measuring latency. COZ can also use progress points to measure the impact of an optimization on latency rather than throughput. To measure latency, a developer must specify two progress points: one at the start of some operation, and the other at the end. The rate of visits to the starting progress point measures the arrival rate, and the difference between the counts at the start and end points tells us how many requests are currently in progress. By denoting L as the number of requests in progress and λ as the arrival rate, we can solve for the average latency W via Little's Law, which holds for nearly any queuing system: $L = \lambda W$. Little[12] Rewriting Little's Law, we then compute the average latency as L/λ.

Little's Law holds under a wide variety of circumstances, and is independent of the distributions of the arrival rate and service time. The key requirement is that Little's Law only holds when the system is *stable*: the arrival rate cannot exceed the service rate. Note that all usable systems are stable: if a system is unstable, its latency will grow without bound.

3.4. Virtual speedup

A critical component of any causal profiler is the ability to virtually speed up any fragment of code. This section describes the derivation of virtual speedup using notation summarized in Table 1. A naive implementation of virtual speedups as shown in Figure 3; each time the function `f` runs, all other threads are paused briefly. If `f` has an average runtime of $\bar{t}_f$ each time it is called and threads are paused for time d each time `f` runs, then `f` has an *effective* average runtime of $\bar{e}_f = \bar{t}_f - d$.

Table 1. Notation used in Section 3.4.

	Table of notation
$\bar{t}_f$	Average runtime of code fragment `f`
d	Length of delay inserted for virtual speedup
n_f	Total number of executions of code fragment `f`
$\bar{e}_f$	Effective average runtime of `f` with virtual speedup
P	Sampling period
s_f	Number of samples in code fragment `f`

If the *real* runtime of `f` was $\bar{t}_f - d$, but we forced every thread in the program to pause for time d after `f` ran (including the thread that just executed `f`) we would measure the same total runtime as with a virtual speedup. The only difference between virtual speedup and a real speedup with these additional pauses is that we use the time d to allow one thread to finish executing `f`. The pauses inserted for virtual speedup increase the total runtime by $n_f \cdot d$, where n_f is the total number of times `f` is called by any thread. Subtracting $n_f \cdot d$ from the total runtime with virtual speedup gives us the execution time we would measure if `f` had runtime $\bar{t}_f - d$.

Implementing virtual speedup with sampling. The previous discussion of virtual speedups assumes an implementation where each execution of small code fragment—a single line of code—causes all other threads instantaneously pause for a fraction of the time require to run the line. Unfortunately, this approach would incur prohibitively high overhead that would distort program execution. Instead, Coz periodically samples the program counter and counts samples that fall in the line selected for virtual speedup. Other threads are delayed proportionally to the number of samples. The number of samples in `f` with a sampling period of P is approximately,

$$s_f \approx \frac{n_f \cdot \bar{t}_f}{P}. \tag{1}$$

In our original model of virtual speedups, delaying other threads by time d each time the selected line is executed has the effect of shortening this line's runtime by d. With sampling, only some executions of the selected line will result in delays. The effective runtime of the selected line *when sampled* is $\bar{t}_f - d$, while executions of the selected line that are not sampled simply take time $\bar{t}_f$. The *effective* average time to run the selected line is,

$$\bar{e}_f = \frac{(n_f - s_f)\cdot \bar{t}_f + s_f \cdot (\bar{t}_f - d)}{n_f}. \tag{2}$$

Using (1), this reduces to,

$$\bar{e}_f = \frac{n_f \cdot \bar{t}_f \cdot \left(1 - \frac{\bar{t}_f}{P}\right) + \frac{n_f \cdot \bar{t}_f}{P} \cdot (\bar{t}_f - d)}{n_f} = \bar{t}_f \cdot \left(1 - \frac{d}{P}\right). \tag{3}$$

The relative difference between t and $\bar{e}_f$, the amount of virtual speedup, is simply,

$$\Delta \bar{t}_f = 1 - \frac{\bar{t}_e}{\bar{t}_f} = \frac{d}{P}. \tag{4}$$

This result lets Coz virtually speed up selected lines without instrumentation. Inserting a delay that is one quarter of the sampling period will virtually speed up the selected line by 25%.

Pausing threads. When one thread receives a sample in the line selected for virtual speedup, all other threads must pause. Rather than using POSIX signals, which would have prohibitively high overhead, Coz controls inter-thread pausing using counters. The first counter, shared by all threads, records the number of times each thread should have paused so far. Each thread has a local counter of the number of times that thread has already paused. Whenever a thread's local count of pauses is less than the number of required pauses in the global counter, a thread must pause (and increment its local counter). To signal all other threads to pause, a thread simply increments both the global counter and its own local counter. Every thread checks if pauses are required after processing its own samples.

Thread creation. To start sampling and adjust delays, Coz intercepts calls to the `pthread_create` function. When a new thread is created, Coz first begins sampling in the new thread. It then inherits the parent thread's local delay count; any previously inserted delays to the parent thread also delayed the creation of the new thread.

Handling suspended threads. Coz only collects samples and inserts delays in a thread while that thread is actually executing. As a result, a backlog of required delays will accumulate in a thread while it is suspended. When a thread is suspended on a blocking I/O operation, this is the desired behavior; pausing the thread while it is already suspended on I/O would not delay the thread's progress. Coz simply adds these delays after the thread unblocks.

However, a thread can also be suspended while waiting for another thread using pthreads synchronization operations. As with blocking I/O, required delays will accumulate while the thread is suspended but Coz may not need to insert all of these delays when the thread resumes. When a thread resumes after waiting on a lock, another thread must have released the lock. If the unlocking thread has executed all the required delays, then the blocked thread has effectively already been delayed. The suspended thread should be credited for any delays inserted in the thread responsible for waking it up. Otherwise, the thread should insert all the necessary delays that accumulated during the time the thread was suspended. To implement this policy, Coz forces threads to execute all required delays before blocking or waking other threads.

Attributing samples to source lines. Samples are attributed to source lines using the source map constructed at startup. When a sample does not fall in any in-scope source line, the profiler walks the sampled callchain to find the first in-scope address. This policy has the effect of attributing all out-of-scope execution to the last in-scope callsite responsible. For example, a program may call `printf`, which calls `vfprintf`, which in turn calls `strlen`. Any samples collected during this chain of calls will be attributed to the source line that issues the original `printf` call.

Optimizations & phase correction. COZ attempts to minimize the number of delays inserted when all threads must be paused. Minimizing delays reduces the performance overhead of profiling without changing the accuracy of virtual speedups. The profile analysis also includes a correction for programs with phases; if left uncorrected, phases could overstate the potential effect of an optimization. For details, see our SOSP paper.[3]

4. EVALUATION

Our evaluation answers the following questions: (1) Does causal profiling enable effective performance tuning? (2) Are COZ's performance predictions accurate?, and (3) Is COZ's overhead low enough to be practical?.

4.1. Experimental setup

We perform all experiments on a 64 core, four socket Advanced Micro Devices (AMD) Opteron machine with 60GB of memory, running Linux 3.14 with no modifications. All applications are compiled using GNU Compiler Collection (GCC) version 4.9.1 at the `-O3` optimization level and debug information generated with `-g`. We disable frame pointer elimination with the `-fno-omit-frame-pointer` flag so Linux can collect accurate call stacks with each sample. COZ is run with the default sampling period of 1ms, with sample processing set to occur after every 10 samples. Each performance experiment runs with a cooling-off period of 10ms after each experiment to allow any remaining samples to be processed before the next experiment begins. Due to space limitations, we only profile throughput (and not latency) in this evaluation.

4.2. Effectiveness

We demonstrate causal profiling's effectiveness through case studies. Using COZ, we collect causal profiles for Memcached, SQLite, and the PARSEC benchmark suite. Using these causal profiles, we were able to make small changes to two of the real applications and six PARSEC benchmarks, resulting in performance improvements as large as 68%. Table 2 summarizes the results of our optimization efforts. We describe some of our experience using COZ here.

Case study: SQLite. The SQLite database library is widely used by many applications to store relational data. This embedded database, which can be included as a single large C file, is used by many applications such as Firefox, Chrome, Safari, Opera, Skype, iTunes, and is a standard component of Android and iOS. We evaluated SQLite performance using a write-intensive parallel workload, where each thread rapidly inserts rows to its own private table. While this benchmark is synthetic, it exposes any scalability bottlenecks in the database engine itself because all threads should theoretically operate independently. We placed a progress point in the benchmark itself (which is linked with the database), which executes after each insertion.

COZ identified three important optimization opportunities, as shown in Figure 4a. At startup, SQLite populates a large number of structs with function pointers to implementation-specific functions, but most of these functions are only ever given a default value determined by compile-time options. The three functions COZ identified unlock a standard pthread mutex, retrieve the next item from a shared page cache, and get the size of an allocated object. These simple functions do very little work, so the overhead of the indirect function call is relatively high. Replacing these indirect calls with direct calls resulted in a 25.60% ± 1.00% speedup.

Comparison with conventional profilers. Unfortunately, running SQLite with gprof segfaults immediately. The application does run with the Linux perf tool, which reports that the three functions COZ identified account for a total of just 0.15% of total runtime (Figure 4b). Using perf, a developer would be misled into thinking that optimizing these functions would be a waste of time. COZ accurately shows that the opposite is true: optimizing these functions has a dramatic impact on performance.

Case study: Memcached. Memcached is a widely-used in-memory caching system. To evaluate cache performance, we ran a benchmark ported from the Redis performance benchmark. This program spawns 50 parallel clients that collectively issue 100,000 `SET` and `GET` requests for randomly chosen keys. We placed a progress point at the end of the `process_command` function, which handles each client request.

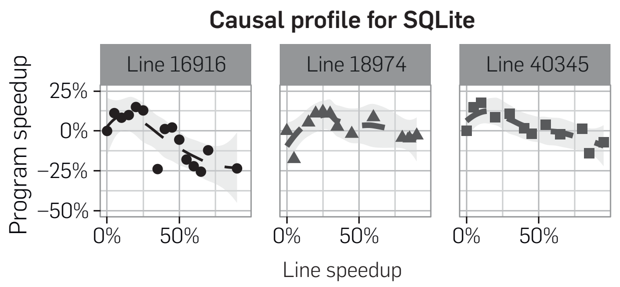
Most of the lines COZ identifies are cases of contention, with a characteristic downward-sloping causal profile plot. One such line is at the start of `item_remove`, which locks an item in the cache and then decrements its reference count, freeing it if the count goes to zero. To reduce lock initialization overhead, Memcached uses a static array of locks to protect items, where each item selects its lock using a hash of its key. Consequently, locking any one item can potentially contend with independent accesses to other items whose keys happen to hash to the same lock index. Reference counts are updated atomically inside this critical section, and the thread that decrements the reference count to zero has exclusive access to this item. As a result, we can safely remove the lock from this function, which yielded a 9.39% ± 0.95% speedup.

Case study: Dedup. The dedup application performs

Table 2. All benchmarks were run 10 times before and after optimization. Standard error for speedup was computed using Efron's bootstrap method. All speedups are statistically significant at the 99.9% confidence level ($\alpha = 0.001$) using the one-tailed Mann-Whitney U test. Diff size reports the number of lines removed and added.

	Summary of optimization results		
Application	Speedup	Diff size	Source lines
Memcached	9.39% ± 0.95%	−6, +2	10,475
SQLite	25.60% ± 1.00%	−7, +7	92,635
blackscholes	2.56% ± 0.41%	−61, +4	342
dedup	8.95% ± 0.27%	−3, +3	2,570
ferret	21.27% ± 0.17%	−4, +4	5,937
fluidanimate	37.5% ± 0.56%	−1, +0	1,015
streamcluster	68.4% ± 1.12%	−1, +0	1,779
swaptions	15.8% ± 1.10%	−10, +16	970

Figure 4. Coz and perf output for SQLite before optimizations. The three lines in the causal profile correspond to the function prologues for `sqlite3MemSize`, `pthreadMutexLeave`, and `pcache1Fetch`. A small optimization to each of these lines will improve program performance, but beyond about a 25% speedup, Coz predicts that the optimization would actually lead to a slowdown. Changing indirect calls into direct calls for these functions improved overall performance by 25.6% ± 1.0%. While the perf profile includes the functions we changed, they account for just 0.15% of the sampled runtime.



(a) Coz's output for SQLite before optimizations.

Perf profile for SQLite

```
% Runtime   Symbol
  85.55%     _raw_spin_lock
     ... 32 lines ...
  0.09%      sqlite3MemSize
   ... 28 lines ...
  0.03%      pcache1Fetch
  0.03%      kmem_cache_free
  0.03%      update_cfs_rq_blocked_load
  0.03%      pthreadMutexLeave
```

(b) Perf's output for SQLite before optimizations.

parallel file compression via deduplication. This process is divided into three main stages: fine-grained fragmentation, hash computation, and compression. We placed a progress point immediately after dedup completes compression of a single block of data (`encoder.c:189`).

Coz identifies the source line `hashtable.c:217` as the best opportunity for optimization. This code is the top of the `while` loop in `hashtable_search` that traverses the linked list of entries that have been assigned to the same hash bucket. These results suggest that dedup's shared hash table has a significant number of collisions. Increasing the hash table size had no effect on performance. This discovery led us to examine dedup's hash function, which could also be responsible for the large number of hash table collisions. We discovered that dedup's hash function maps keys to just 2.3% of the available buckets; over 97% of buckets were never used during the entire execution.

The original hash function adds characters of the hash table key, which leads to virtually no high order bits being set. The resulting hash output is then passed to a bit shifting procedure intended to compensate for poor hash functions. We removed the bit shifting step, which increased hash table utilization to 54.4%. We then changed the hash function to bitwise XOR 32 bit chunks of the key. Our new hash function increased hash table utilization to 82.0% and resulted in an 8.95% ± 0.27% performance improvement. The entire optimization required changing just three lines of code, and the entire profiling and tuning effort took just two hours.

Comparison with gprof. We ran both the original and optimized versions of dedup with gprof. The optimization opportunities identified by Coz were not obvious in gprof's output. Overall, `hashtable_search` had the largest share of highest execution time at 14.38%, but calls to `hashtable_search` from the hash computation stage accounted for just 0.48% of execution time; Gprof's call graph actually obscured the importance of this code. After optimization, `hashtable_search`'s share of execution time reduced to 1.1%.

Case study: Fluidanimate and streamcluster. The fluidanimate and streamcluster benchmarks use a similar approach to parallelism. The application spawns worker threads that execute in a series of concurrent phases, with each phase separated from the next by a barrier. We placed a progress point immediately after the barrier, so it executes each time all threads complete a phase of the computation.

In both benchmarks, Coz also identified the same points of contention: PARSEC's barrier implementation. Figure 5 shows the evidence for this contention in fluidanimate. This profile tells us that optimizing the indicated line of code would actually *slow down* the program, rather than speed it up. Both lines run immediately before entering a loop that repeatedly calls `pthread_mutex_trylock`. Removing this spinning from the barrier would reduce contention, but it was easier to replace the custom barrier with the default `pthread_barrier` implementation. This one line change led to a 37.5% ± 0.56% speedup in fluidanimate, and a 68.4% ± 1.12% speedup in streamcluster.

Effectiveness Summary. Our case studies confirm that Coz is effective at identifying optimization opportunities and guiding performance tuning. In every case, the information Coz provided led us directly to the optimization we implemented. In most cases, Coz identified around 20 lines of interest, with as many as 50 for larger programs (Memcached and x264). Coz identified optimization opportunities in all of the PARSEC benchmarks, but some required more invasive changes that are out of scope for this paper.

4.3. Accuracy

For most of the optimizations described above, it is not possible to quantify the effect our optimization had on the specific lines that Coz identified. However, for two of our case studies—ferret and dedup—we can directly compute the effect our optimization had on the line Coz identified and compare the resulting speedup to Coz's predictions. Our results show that Coz's predictions are highly accurate.

For dedup, Coz identified the top of the `while` loop that traverses a hash bucket's linked list. By replacing the degenerate hash function, we reduced the average number of elements in each hash bucket from 76.7 to 2.09. This change reduces the number of iterations from 77.7 to 3.09 (accounting for the final trip through the loop). This reduction

Figure 5. Coz output for fluidanimate, prior to optimization. Coz finds evidence of contention in two lines in `parsec_barrier.cpp`, the custom barrier implementation used by both fluidanimate and stream-cluster. This causal profile reports that optimizing either line will slow down the application, not speed it up. These lines precede calls to `pthread_mutex_trylock` on a contended mutex. Optimizing this code would increase contention on the mutex and interfere with the application's progress. Replacing this inefficient barrier implementation sped up fluidanimate and streamcluster by 37.5% and 68.4% respectively.

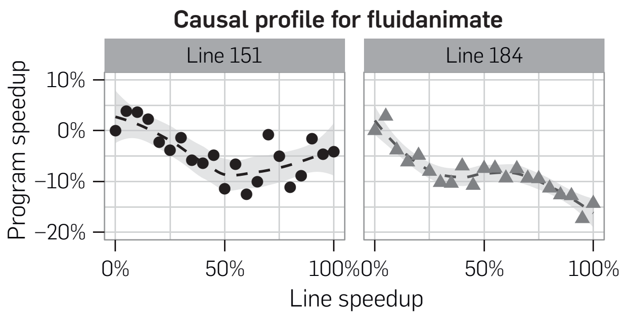
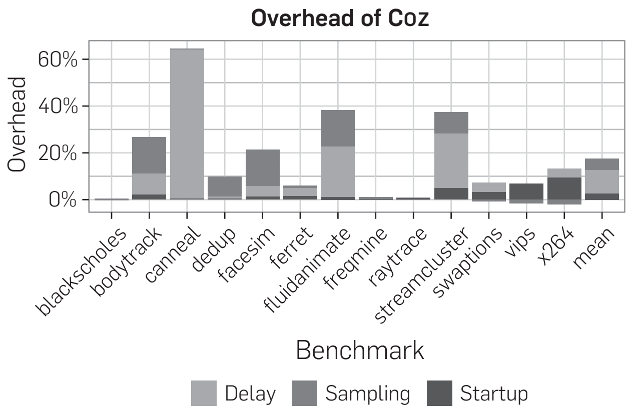


Figure 6. Percent overhead for each of Coz's possible sources of overhead. *Delay* is the overhead from adding delays for virtual speedups, *Sampling* is the cost of collecting and processing samples, and *Startup* is the initial cost of processing debugging information. Note that sampling results in slight performance *improvements* for swaptions, vips, and x264.



corresponds to a speedup of the line Coz identified by 96%. For this speedup, Coz predicted a performance improvement of 9%, very close to our observed speedup of 8.95%. Results for ferret are similar; Coz predicted a speedup of 21.4%, and we observe an actual speedup of 21.2%.

4.4. Efficiency

We measure Coz's profiling overhead on the PARSEC benchmarks running with the native inputs. The sole exception is streamcluster, where we use the test inputs because execution time was excessive with the native inputs.

Figure 6 breaks down the total overhead of running Coz on each of the PARSEC benchmarks by category. The average overhead with Coz is 17.6%. Coz collects debug information at startup, which contributes 2.6% to the average overhead. Sampling during program execution and attributing these samples to lines using debug information is responsible for 4.8% of the average overhead. The remaining overhead (10.2%) comes from the delays Coz inserts to perform virtual speedups.

These results were collected by running each benchmark in four configurations. First, each program was run without Coz to measure a baseline execution time. In the second configuration, each program was run with Coz, but execution terminated immediately after startup work was completed. Third, programs were run with Coz configured to sample the program's execution but not to insert delays (effectively testing only virtual speedups of size zero). Finally, each program was run with Coz fully enabled. The difference in execution time between each successive configuration give us the startup, sampling, and delay overheads, respectively.

Reducing overhead. Most programs have sufficiently long running times (mean: 103s) to amortize the cost of processing debug information, but especially large executables can be expensive to process at startup (e.g., `x264` and `vips`). Coz could be modified to collect and process debug information lazily to reduce startup overhead. Sampling overhead comes mainly from starting and stopping sampling with the `perf_event` API at thread creation and exit. This cost could be amortized by sampling globally instead of per-thread, which would require root permissions on most machines. If the `perf_event` API supported sampling all threads in a process, this overhead could be eliminated. Delay overhead, the largest component of Coz's total overhead, could be reduced by allowing programs to execute normally for some time between each experiment. Increasing the time between experiments would significantly reduce overhead, but a longer profiling run would be required to collect a usable profile.

Efficiency summary. Coz's profiling overhead is on average 17.6% (minimum: 0.1%, maximum: 65%). For all but three of the benchmarks, its overhead was under 30%. Given that the widely used gprof profiler can impose much higher overhead (e.g., 6 times for ferret, versus 6% with Coz), these results confirm that Coz has sufficiently low overhead to be used in practice.

5. RELATED WORK

Causal profiling differs from past profiling techniques, which have focused primarily on collecting as much detailed information as possible about a program without disturbing its execution. Profilers have used a wide variety of techniques to gather different types of information in different settings, which we summarize here.

5.1. General-purpose profilers

General-purpose profilers are designed to monitor where a program spends its execution time. Profilers such as gprof and oprofile are typical of this category.[7, 11] While oprofile uses sampling exclusively, gprof mixes sampling and instrumentation to measure both execution time and collect call graphs, which show how often each function was called, and where it was called from. Later extensions to this work have reduced the overhead of call graph profiling and added additional detail with path profiling, but the functionality of general-purpose profilers remains largely unchanged. This kind of profiling information is useful for identifying where a program spends its time, but not necessarily where developers should work to improve performance.

5.2. Parallel profilers

Several techniques have been used to identify performance and scalability bottlenecks in parallel programs. Systems such as IPS trace the execution of a running program to identify its *critical path*, the longest sequence of dependencies in the complete program dependence graph.[13] While this approach can work well for message-passing systems, it would require instrumenting all memory accesses in a modern shared-memory parallel program; this would impose substantial overhead, likely distorting the results far too much to be representative of an un-profiled execution.

Other parallel profilers, such as FreeLunch and the WAIT tool, identify code that runs while some of a program's threads sit idle.[1, 4] These systems assign some level of blame for blocking to all of a program's code. The idea is that code running while other threads are blocked must be responsible for the reduced parallelism. This heuristic works well for some parallel performance issues, but not all performance bottlenecks change a thread's scheduler state.

5.3. Profiling for scalability

Several systems have been developed to measure potential parallelism in serial programs.[6, 16, 17] Other systems instead examine parallel programs to predict how well the program will scale to larger numbers of hardware threads.[10] These approaches are distinct and complimentary to causal profiling. These tools help developers parallelize and scale applications, while COZ helps developers improve an existing parallel program at the current level of parallelism.

5.4. Performance experimentation

COZ is a significant departure from past profiling techniques in that it intentionally perturbs a program's execution to model the effect of an optimization. While this technique is unique for software profilers, the idea of a performance experiment has appeared in other systems. Mytkowicz et al.[14] use delays to validate the output of profilers on single-threaded Java programs.[14] Snelick et al.[15] use delays to profile parallel programs.[15] This approach measures the effect of slowdowns in combination, which requires a complete execution of the program for each of an exponential number of configurations. While these techniques involve performance experiments, COZ is the first system to use performance perturbations to create the effect of an optimization.

6. CONCLUSION

Profilers are the primary tool in the programmer's toolbox for identifying performance tuning opportunities. Previous profilers only observe actual executions and correlate code with execution time or performance counters. This information can be of limited use because the amount of time spent does not necessarily correspond to where programmers should focus their optimization efforts. Past profilers are also limited to reporting end-to-end execution time, an unimportant quantity for servers and interactive applications whose key metrics of interest are throughput and latency. Causal profiling is a new, experiment-based approach that establishes causal relationships between hypothetical optimizations and their effects. By virtually speeding up lines of code, causal profiling identifies and quantifies the impact on either throughput or latency of any degree of optimization to any line of code. Our prototype causal profiler, COZ, is efficient, accurate, and effective at guiding optimization efforts. Coz is now a standard package on current Debian and Ubuntu platforms and can be installed via the command `sudo apt-get install coz-profiler` or it can be installed from source on any Linux distribution; all source is at http://coz-profiler.org.

Acknowledgments

This material is based upon work supported by the National Science Foundation under Grants No. CCF-1012195 and CCF-1439008. Charlie Curtsinger was supported by a Google PhD Research Fellowship. The authors thank Dan Barowy, Steve Freund, Emma Tosch, John Vilk, and Tim Harris for their feedback and helpful comments.

References

1. Altman, E.R., Arnold, M., Fink, S., Mitchell, N. Performance analysis of idle programs. *OOPSLA. ACM* (2010), 739–753.
2. Curtsinger, C., Berger, E.D. Stabilizer: Statistically sound performance evaluation. In *ASPLOS* (New York, NY, USA, 2013), ACM.
3. Curtsinger, C., Berger, E.D. COZ: Finding code that counts with causal profiling. In *SOSP*, (ACM, New York, NY, 2015), 184–197.
4. David, F., Thomas, G., Lawall, J., Muller, G. Continuously measuring critical section pressure with the free-lunch profiler. In *OOPSLA*, (ACM, New York, NY, 2014), 291–307.
5. Free Software Foundation. *Debugging with GDB*, 10th edn., The Free Software Foundation, Boston, MA.
6. Garcia, S., Jeon, D., Louie, C.M., Taylor, M.B. Kremlin: rethinking and rebooting gprof for the multicore age. In *PLDI*, (ACM, New York, NY, 2011), 458–469.
7. Graham, S.L., Kessler, P.B., McKusick, M.K. Gprof: A call graph execution profiler. In *SIGPLAN Symposium on Compiler Construction*, (ACM, New York, NY, 1982), 120–126.
8. Intel. *Intel VTune Amplifier*, 2015.
9. kernel.org. *perf: Linux profiling with performance counters*, 2014.
10. Kulkarni, M., Pai, V.S., Schuff, D.L. Towards architecture independent metrics for multicore performance analysis. *SIGMETRICS Performance Evaluation Review 38*, 3 (2010), 10–14.
11. Levon, J., Elie, P. Oprofile: A system profiler for Linux, 2004. http://oprofile.sourceforge.net/.
12. Little, J.D. OR FORUM: Little's Law as Viewed on Its 50th Anniversary. *Operations Research 59*, 3 (2011), 536–549.
13. Miller, B.P., Yang, C.-Q. IPS: An interactive and automatic performance measurement tool for parallel and distributed programs. In *ICDCS*, 1987, 482–489.
14. Mytkowicz, T., Diwan, A., Hauswirth, M., Sweeney, P.F. Evaluating the accuracy of Java profilers. In *PLDI* (2010) ACM, 187–197.
15. Snelick, R., JáJá, J., Kacker, R., Lyon, G. Synthetic-perturbation techniques for screening shared memory programs. *Software Practice & Experience 24*, 8 (1994), 679–701.
16. von Praun, C., Bordawekar, R., Cascaval, C. Modeling optimistic concurrency using quantitative dependence analysis. In *PPoPP* (2008), ACM, 185–196.
17. Zhang, X., Navabi, A., Jagannathan, S. Alchemist: A transparent dependence distance profiling infrastructure. In *CGO* (2009), IEEE Computer Society, 47–58.

Charlie Curtsinger (curtsinger@grinnell.edu), Department of Computer Science, Grinnell College, USA.

Emery D. Berger (emery@cs.umass.edu), College of Information and Computer Sciences, University of Massachusetts Amherst, USA.

King Abdullah University of Science and Technology

Faculty Positions: Computer, Electrical and Mathematical Science and Engineering

Division: Computer Science and Electrical Engineering Faculty Positions in Computer Science 2018

Location: KAUST, Thuwal, Kingdom of Saudi Arabia

The Computer, Electrical, and Mathematical Sciences and Engineering Division (https://cemse.kaust.edu.sa) at King Abdullah University of Science and Technology (KAUST) invites applications for faculty positions at all levels (Full, Associate, and Assistant Professor) in the following areas of Computer Science (https://cs.kaust.edu.sa):

▸ Data Mining and analytics with the emphasis on Big Data
▸ Machine Learning with emphasis on Deep Learning
▸ Artificial intelligence
▸ Data Science with emphasis on Statistical methods
▸ Computer Systems and Emerging Architectures (GPU's, FPGA's, etc)
▸ High performance computing
▸ Computer security
▸ Quantum Computing

High priority will be given to the overall originality and promise of the candidate's work rather than the candidate's sub-area of specialization within Computer Science. An earned PhD in Computer Science, Computer Engineering, Electrical Engineering, or a related field, and strong publication record in top-tier conferences and journals, are required. Senior candidates must have demonstrated strong leadership in the field. Women are strongly encouraged to apply. Also of particular interest are interdisciplinary candidates who can open new areas of investigation. A successful candidate will be expected to teach courses at the graduate level and to build and lead a research group of postdoctoral fellows and graduate students. Faculty members enjoy secure research funding from KAUST and have opportunities for additional funding through several KAUST provided sources and through industry collaborations.

KAUST (https://www.kaust.edu.sa) is an international, graduate-only research university, located on the shores of the Red Sea in Saudi Arabia. KAUST offers superb research facilities, including the 5 Petaflop/s Shaheen-2 supercomputer and a large-scale virtual reality installation; generous assured research funding; and internationally competitive salaries. KAUST attracts top international faculty, scientists, engineers and students to conduct curiosity-driven and goal-oriented research to address the world's pressing scientific and technological challenges.

Apply online at https://kaust.link/apply-faculty-Computer-Science-2018

KIT
Karlsruher Institut für Technologie

Karlsruhe Institute of Technology (KIT) – The Research University in the Helmholtz Association creates and imparts knowledge for the society and the environment. It is our goal to make significant contributions to mastering the global challenges of mankind in the fields of energy, mobility, and information. For this, about 9300 employees of KIT cooperate in a broad range of disciplines in research, academic education, and innovation.

The Institute of Theoretical Informatics within the Division II – Informatics, Economics, and Society – is seeking to fill, as soon as possible, the position of a

W3-Professorship in Algorithmics

The research area of the professorship is supposed to be part of the core area of algorithmics. In particular, both algorithm theory and algorithm engineering are welcome. The position holder is expected to contribute to KIT's strategic focus on „Algorithm Engineering for the Scalability Challenge" and to engage appropriately in teaching of the courses of study in informatics at the KIT Department of Informatics, both in the compulsory and elective areas (in German and English language). There is a transitional period for acquiring German language skills.

The KIT is looking for a person with excellent scientific qualifications and an excellent international track record in the corresponding field of research. Extensive experience in acquiring third-party funds and leading scientific working groups is expected, as well as very good didactic skills and several years of experience and high commitment in independent teaching, both in basic informatics lectures and in advanced courses on topics from the research area of the professorship.

In all other respects, the requirements for appointment are in accordance with the Higher Education Act of the state of Baden-Wuerttemberg (LHG), Article 47.

KIT is pursuing a gender equality policy and encourages women to apply. Furthermore, in case of equal qualification, preference is given to applicants with disabilities.

Applications with the usual documents (CV, credentials, academic background and list of publications) and a perspective paper of no more than five pages on the planned research and teaching activities as well as (possibly interdisciplinary) collaborations at KIT must be submitted by **September 14th, 2018** to the **Karlsruhe Institute of Technology, Dean of the KIT Department of Informatics, Am Fasanengarten 5, D-76131 Karlsruhe, email: dekanat@informatik.kit.edu.** For additional information, please contact Prof. Dr. Peter Sanders, Tel.: +49 721/608-43985; email: peter.sanders@kit.edu.

Further details can be found on our website: **www.kit.edu**.

KIT – The Research University in the Helmholtz Association

Yeshiva University

Assistant Professor of Computer Science
Stern College for Women

Are you committed to Computer Science education for women?

Stern College for Women, a selective private college in the heart of "Silicon Alley", in midtown Manhattan, New York, seeks a full-time Assistant Professor of Computer Science. This is an excellent opportunity to contribute to the growth of Computer Science education for women, in a women-only program. Our students are enthusiastic learners, CS enrollment is growing, and our graduates are being sought by premiere technology and finance employers.

A Ph.D. in Computer Science or a related field is required by the start of employment, along with a successful teaching record or distinguished industrial experience. The successful candidate for this tenure-track position will teach a range of undergraduate courses leading to a Bachelor's degree in Computer Science, and will pursue a research program that encourages students to participate in the discovery process. Applicants should submit, by PDF email attachment: a cover letter, CV, teaching statement, research statement, and three letters of recommendation.

To apply for this position, view Job ID 2017-9926 at www.yu.edu/careers.

Questions about this position should be directed to alan.broder@yu.edu.

Skills and Competencies:
- Excellent communication skills, with a strong commitment to teaching
- Demonstrated ability to mentor and inspire
- Research or advanced industrial experience in computational biology, security & privacy, network science, data science, or AI
- Women and minorities are strongly encouraged to apply

About Us:
Stern College for Women is a part of Yeshiva University (YU), founded in 1886. YU has a strong tradition of combining Jewish scholarship with academic excellence and achievement in the liberal arts, sciences, medicine, law, business, social work, Jewish studies, education, psychology, and more.

We are a leading global educational institution that employs 2000 people across our various campuses. We seek to attract and retain engaged and committed individuals who contribute to an exciting working environment, where there is a sense of community and belonging, balanced with a significant cross section of people from diverse backgrounds working and studying together. The University offers an excellent compensation package, and employee benefit plans.

Yeshiva University is an equal opportunity employer committed to hiring minorities, women, individuals with disabilities and protected veterans.

John L. Hennessy

[CONTINUED FROM P. 104] project, but you'll never get these ideas to work in real machines that people want to sell to customers."

What were the most common objections?

JOHN: We built academic prototypes, and they didn't have everything you'd need for a commercial machine. And people said, "When you put in virtual memory support, or support for floating point, all the advantages you have will disappear."

DAVID: The conventional wisdom at the time was software was buggy because the vocabulary of the computers it talked to was too low. So there were all these efforts to try and make the vocabulary closer to that of the languages. John and I were arguing the opposite, and that was part of the heresy.

JOHN: I think the other thing that undermined it was that we didn't have a good scientific explanation of what was happening until later. That was part of our motivation to write the book, so that we could explain the ideas quantitatively and scientifically.

You're referring to *Computer Architecture: A Quantitative Approach*, which was first published in 1989 and is now in its sixth edition.

DAVID: I think a lot of faculty are unhappy with the textbooks they use. John and I talked about it at meetings, and when I realized I was going to be the next chair of the computer science division at Berkeley, we decided to go for it, because that gave us a deadline.

You handed off chapters to one another via FedExed floppy disks.

DAVID: We prototyped the book as if it were a computer, with an alpha and a beta version.

Microsoft's David Cutler bought copies for all members of his design team.

DAVID: I think Microsoft also kept a copy in the stationery store. Pads of paper, pencils, Hennessy and Patterson…

JOHN: In the first year, we probably sold as many books to practicing engineers as we did to the academics. That's really unusual. Books usually divide one way or the other—either they're written for professionals or for the university market.

DAVID: The book made those ideas accessible to lots of people.

You also invented a brand new parameterized architecture for the book called DLX that expressed your approach.

DAVID: There are lots of RISC instruction sets, and John is associated with one and I'm associated with another. From a textbook writer's perspective, we thought that picking one might flavor the book, so we decided to invent a brand-new instruction set.

MIPS lost out to Intel in the PC market, but RISC processors now power nearly

all smartphones and mobile devices.

JOHN: The key thing to understand is the efficiency issue. When Dave and I were developing RISC architectures, we had to find efficient ways to use the silicon that was available at the time to get the highest-performance machines. Today, you're constrained by a whole set of different factors. Sometimes it's a question of the silicon area, because you've got to be able to sell processors for a dollar. But the other big constraint is energy efficiency, because so many things are battery powered, or you can't include a fan to cool them. So the RISC's underlying efficiency has enabled it to catapult into this critical role.

DAVID: It's kind of like "Back to The Future." When they're selling things for pennies, people care a lot about the number of transistors they use. At the very high end, the instruction set matters less because there are so many other things going on. At the low end, you even worry about how many registers you have. What's nice about RISC is that it works fine at the high end, and it's a big asset at the low end, and that's why it's been so successful.

What excites you both in the field of computer architecture?

JOHN: As conventional uniprocessors stall out, there's a focus now on what Dave and I have called domain-specific architectures—architectures that are designed for specific classes of problems. The obvious example now is deep neural networks; they use very specific computing strategies, and you can get an order of magnitude in efficiency by designing an architecture that does the kinds of functions that these machines need to do well.

"What's nice about RISC is that it works fine at the high end, and it's a big asset at the low end, and that's why it's been so successful."

David A. Patterson

DAVID: The reason that John and I are such good co-authors is that we have almost identical world views. Domain-specific architectures are in the newest chapter of our book, and for sure they're an exciting development.

There are two other things I'm interested in. First is RISC-V, an open instruction set architecture that's based on RISC principles. Not too many people get to work on proprietary instruction sets like ARM and x86, but everybody can get involved in the instruction set evolution of RISC-V. The other thing is getting better at security. So far, we haven't asked much of computer hardware in security. I think architects need to step up and really help attack this problem. What's exciting about the RISC-V is that you can download a full viable software stack, prototype your idea using an FPGA, stick it on the Internet, let people attack it, and see whether or not it works. The iteration loop can be days instead of the years it takes with proprietary instruction sets.

John, you're also involved with the Knight-Hennessy Scholars Program, which aims to "build a multidisciplinary community of Stanford graduate students dedicated to finding creative solutions to the world's greatest challenges."

JOHN: We just admitted our first class coming this fall. We have 49 students from 35 different countries. They have already accomplished amazing things, and they represent every school in the university, from law and business to engineering and the social sciences. Given the gigantic leadership problems that we have around the world, hopefully we can get a new generation of young people out there that are determined to do better.

John, you're also writing a book about leadership.

JOHN: It is at the publisher now. It's a short book—it's meant to be readable on a cross-country flight. It's about my experiences with a variety of leadership issues and what I learned about it. The first chapter is humility.

Anything else?

DAVE: With the ending of Dennard Scaling and Moore's Law, we must change the instruction set for major gains in cost, performance, energy, or security. Freeing architects from the chains of proprietary instruction sets may spur innovation like in the 1980s. Hence the title of our Turing Lecture: "A New Golden Age for Computer Architecture: Domain Specific Hardware/Software Co-Design, Enhanced Security, Open Instruction Sets, and Agile Chip Development."

Leah Hoffmann is a technology writer based in Piermont, NY, USA.

DOI:10.1145/3204453

Leah Hoffmann

Q&A RISC Management

ACM A.M. Turing Award recipients John Hennessy and David Patterson have introduced generations of students to Reduced Instruction Set Computing.

AT A TIME when "making an impact" can feel like a vague or even overwhelming prospect, it's worth reviewing the accomplishments of two scientists who have done just that: ACM A.M. Turing Award recipients John Hennessy and David Patterson. What began as a simple-sounding insight—that you could improve microprocessor performance by including only instructions that are actually used—blossomed into a paradigm shift as the two honed their ideas in the MIPS (Microprocessor without Interlocked Pipeline Stages) and RISC (Reduced Instruction Set Computer) processors, respectively. A subsequent textbook, *Computer Architecture: A Quantitative Approach*, introduced generations of students not just to that particular architecture, but to critical principles that continue to guide designers as they balance constraints and strive for maximum efficiency.

Patterson (left) and Hennessy on the campus of Stanford University.

David, you began working on what became the RISC architecture after a leave of absence at Digital Equipment Corporation (DEC).

DAVID PATTERSON: My sabbatical at DEC focused on reducing microprogramming bugs. At the time, people were designing microprocessors to imitate minicomputers like the ones in DEC's popular VAX family, but VAX had an extremely complicated instruction set. When I got back to Berkeley, I wrote a paper arguing that if you put that into silicon, you need to make a chip that has a repair mechanism, because of all the bugs.

That paper was rejected.

DAVID: It's a dumb way to design microprocessors if you have to repair microcode bugs. That got me thinking about how to build something simpler that made more sense for microprocessors.

So during a series of graduate courses that began in 1980, you began developing a fast, lean microprocessor that included only instructions that were actually used. How did you meet John?

DAVID: There was a (U.S. Defense Advanced Research Projects Agency) DARPA meeting at the (University of California,) Berkeley campus in May of 1980, and I was presenting some of our early results. And there was this young "graduate student" from Stanford (University)—do you remember what you were presenting then, John?

JOHN HENNESSY: It was probably the microcode compiler that I wrote that got used for (James H.) Clark's Geometry Engine project.

John, you went back to Stanford and, in 1981, started working on your own Reduced Instruction Set microprocessor, which you eventually commercialized as the Microprocessor without Interlocked Pipeline Stages, or MIPS. However, the ideas behind RISC and MIPS were controversial at first.

JOHN: People said, "This is fine for an academic [CONTINUED ON P. 102]

Call for Papers and Sponsors

VRST2018

24th ACM Symposium on Virtual Reality Software and Technology

Nov. 28th - Dec. 1st, 2018, Tokyo, Japan

https://vrst.acm.org/vrst2018/

The ACM Symposium on Virtual Reality Software and Technology (VRST) is the premier international symposium for the presentation of new research results, systems and techniques among researchers and developers concerned with augmented, virtual and mixed reality (AR/VR/MR) software and technology.

The VRST brings together the main international research groups working on AR/VR/MR, along with many of the world's leading companies that provide or consume AR/VR systems. The VRST 2018 conference will be held in Tokyo, Japan, hosted by Waseda University, from Wednesday, Nov. 28th to Saturday, Dec. 1st, 2018.

Submission of Papers

Authors are invited to submit papers of no more than 10 pages for full-paper and 4 pages for short-paper with 2-column "teaser" figures on the front page. Papers and poster abstracts are should be prepared with the "sigconf" ACM template style. The ACM article template packages (LaTeX and Word) are available from: http://www.acm.org/publications/proceedings-template

Extended versions of two or three best papers from VRST 2018 will be invited to IEEE Transactions on Visualization and Computer Graphics.

Conference Topics

VRST 2018 welcomes submissions of research papers that relate (but not limited) to topics given below.

- VR/AR/MR(=XR) technology and devices
- Advanced display technologies and immersive projection technologies
- Low-latency and high-performance XR
- Multi-user and distributed XR
- XR software infrastructures
- XR authoring systems
- Human interaction and collaborative techniques for XR
- Input devices for XR
- Tracking and sensing
- Multisensory and multimodal system for XR
- Haptics, smell and taste
- Audio and music processing for XR
- Brain-computer interfaces
- Computer graphics techniques for XR
- Computer vision techniques for XR
- Modeling and simulation
- Real-time rendering
- Real-time physics-based modeling
- XR applications (e.g. training systems, medical systems, serious games...)
- Avatars and virtual humans in XR
- Tele-operation and telepresence
- Performance testing & evaluation
- Multi-user and distributed XR
- Locomotion and navigation in virtual environments
- Perception, presence, virtual embodiment, and cognition
- Teleoperation and telepresence
- Ethical issues in XR
- Physically based rendering for XR
- Computer animation for XR
- Sound synthesis for XR
- XR for fabrication

Call for Exhibitors and Sponsors

The VRST 2018 offers exhibitors and sponsors an opportunity to showcase their company's products, attractive demos and innovative solutions at the symposium. Please refer to the conference web-page for information about singing up to become an exhibitor or sponsor at VRST 2018.

Important Dates

August 15, 2018 (23:59 PST)
Papers with all material submission deadline

September 1, 2018 (23:59 PST)
Posters and demos submission deadline

September 25, 2018
Author notification for papers, posters and demos

October 1, 2018
Camera-ready papers, posters, and demos due

November 28 - December 1, 2018
Conference in Tokyo, Japan

Steering Committee Chair

Yoshifumi Kitamura (Tohoku University)

Symposium General Chair

Shigeo Morishima (Waseda University)

Technical Program Chairs

Yuichi Itoh (Osaka University)
Rob Lindeman (University of Canterbury)
Takaaki Shiratori (Facebook)
Yonghao Yue (University of Tokyo)

Poster and Demos Chairs

Takashi Ijiri (Shibaura Institute of Tech.)
Hideki Koike (TITECH)
Hideki Todo (Chuo Gakuin University)
Hubert Shum (Northumbria University)

acm

SIGCHI

ACMSIGGRAPH

NICT

WASEDA University

s2018.siggraph.org

The 45th International Conference & Exhibition on Computer Graphics & Interactive Techniques

Sponsored by ACMSIGGRAPH

GENERATIONS / **VANCOUVER** 12-16 AUGUST

SIGGRAPH2018

REGISTER TODAY TO SAVE

BEST SAVINGS BY 22 JUNE