COMMUNICATIONS OF THE ACM

CACM.ACM.ORG

09/2019 VOL.62 NO.09

Computational Sustainability

Computing for a Better World and a Sustainable Future

Augmented Reality Gets Real

Alloy: A Language and Tool for Exploring Software Designs

Sustaining Open Collaboration in Universities

Association for Computing Machinery

2019 the year of anniversaries

Time to reflect and look ahead

/ 30th anniversary of the fall of the Berlin Wall
/ 30th ACM Hypertext Conference
/ 30th anniversary of the WWW
/ 20th anniversary of the first ACM Hypertext Conference in Germany

Scope

The ACM Hypertext conference is a premium venue for high quality peer-reviewed research on hypertext theory, systems, and applications. It is concerned with all aspects of modern hypertext research including social media, semantic web, dynamic and computed hypertext and hypermedia as well as narrative systems, and applications. The theme of HT'19 is **HYPERTEXT – TEAR DOWN THE WALL.**

Organization

General Chairs

Claus Atzenbeck
Hof University, Germany
@clausatz

Jessica Rubart
OWL University, Germany
@jrubart

Program Chair

David Millard
University of Southampton, UK
@hoosfoos

join in and get in touch with us

Subscribe to our HT'19 channels for live updates on keynotes, scientific talks, social events, interviews, pictures, and many great impressions.

human.iisys.de/ht2019
facebook.com/acmht
@ACMHT

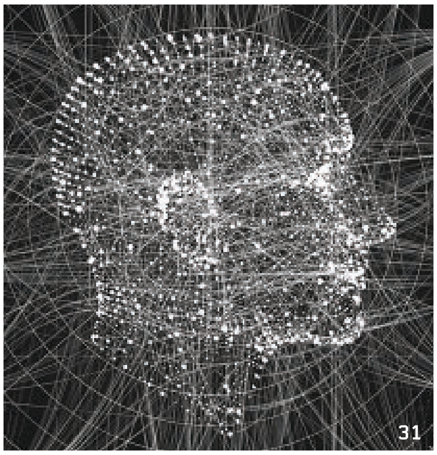
30th Anniversary ACM Conference on Hypertext and Social Media

Sept. 17–20, 2019

Hof University, Germany

acm sigweb hof university iisys

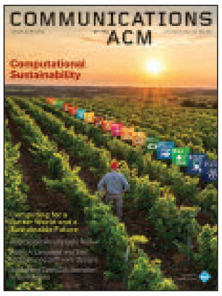
COMMUNICATIONS OF THE ACM



IMAGES BY: (L) PRITSANA; (R) GARRY KILLIAN

09/2019
VOL. 62 NO. 09

IMAGES BY: (L) LEUNG CHO PAN; (R) IDEA STUDIO



About the Cover:
Computer and information researchers combine efforts with other scientific disciplines in an extraordinary pursuit: to solve the mounting challenges of a sustainable future for all of humanity. Their story begins on p. 56. Cover composition by Andrij Borys Associates; photo by Jack Frog.

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, 1601 Broadway, 10th Floor New York, NY 10019-7434 USA. Periodicals postage paid at New York, NY 10001, and other mailing offices.

POSTMASTER
Please send address changes to *Communications of the ACM*
1601 Broadway, 10th Floor
New York, NY 10019-7434 USA

Printed in the USA.

Association for Computing Machinery

PLEASE RECYCLE THIS MAGAZINE

DOI:10.1145/3354460

Andrew A. Chien

Sustaining Open Collaboration in Universities

CORE UNIVERSITY MISSIONS of scholarly education, the advancement of knowledge, and the rigorous evaluation of ideas make universities bastions of open collaboration. In computing, such open sharing has fueled computing's rapid advance and created win-win-win global partnerships in education, innovation, and use, benefitting all. But increasing international tension and distrust,[a] and its projection into universities, is eroding open collaboration and inquiry.[b]

Trade secret and intellectual property walls. In 1980, the Bayh-Dole Act recast universities as intellectual property owners raising new barriers to information sharing in universities.[c] Technology and trade secrets under non-disclosure agreement (NDA). Exclusive licensing. Competing with industry partners. Billion-dollar litigation. These practices force faculty, staff, and students to self-censor communication: "Have you signed an NDA?" "Have they licensed technology X? "Is person Y consulting at startup Z?" By 2000, an inspired response sought to create "demilitarized zones" for intellectual property.[d] As Intel VP of Research, I was stunned by a young professor's decrial of the situation: "Why do I have to understand intellectual property contracts, NDAs, and keep secrets? I became an academic to create new knowledge and share it, not to become an expert in IP law."

In the past two decades, U.S. universities have experienced significant growth in foreign-national students at both undergraduate and graduate levels (now 65% in computing Ph.D. programs). Drawn by U.S. leadership in computing, and the opportunities of an open university educational and research environment, their contributions to the U.S. economy are well documented.[e] International student exchange has benefitted individuals, economies, and nations.

National boundary walls. We are experiencing growing international tension, arising from competition for economic leadership, regional influence and hegemony, and even military advantage, often between political systems (democracy, autocracy).[f] The international diversity of university communities translates national sovereignty over citizens directly into boundaries in the university—departments, research groups, or even classrooms. Broadening definitions of sensitive technology imposed by sovereign governments raises new walls. Increasing restrictions on knowledge/technology sharing and interaction with untrusted individuals and entities raise new walls. Interaction restrictions are particularly corrosive—compliance can require off-putting questions—"What is your citizenship?" "Who is your employer?" "With whom, and under what circumstances, can this knowledge be shared?" Recent faculty guidance at a leading university allowed "speaking with employees of X, but only if they are U.S. citizens."[g] A standing government directive requires faculty to vet all new collaborations with a list of untrusted individuals and entities.[h]

Fewer walls. Shaping less damaging walls. What to do? Walls arise from different forces. We have learned to shape those that arise from the profit motive, shaping them to balance inquiry with entrepreneurship. Walls that arise from national security and sovereignty cannot be resisted; so perhaps, in analogous fashion, we can shape them to minimize damage to open collaboration.

To this end, faculty should engage and shape policies to limit the harm of rising barriers and defend university missions of education, invention, and rigorous evaluation. This spirited defense must be in the cause of society—not perceived as a parochial "academic" or research community interest.

The ACM and IEEE Computer Society, as international leadership organizations, must work to shape national policies around the world to support open collaboration and inquiry in universities.

Andrew A. Chien, EDITOR-IN-CHIEF

Andrew A. Chien is the William Eckhardt Distinguished Service Professor in the Department of Computer Science at the University of Chicago, Director of the CERES Center for Unstoppable Computing, and a Senior Scientist at Argonne National Laboratory.

a Chien, A. Open collaboration in an age of distrust. *Commun. ACM* (Jan. 2019).

b I focus on U.S. universities, but those in many countries share these ideals.

c National Research Council. *Managing University Intellectual Property in the Public Interest.* The National Academies Press, 2011.

d Tennenhouse, D. Intel's open collaborative model of industry-university research. *J. RTM*, 2004.

e Anderson, S. Immigrants and Billion-Dollar Companies. National Foundation for American Policy, Oct. 2018.

f Hong Kong's violent protests against Chinese rule. *Economist* (July 27, 2019).

g Lee, J.L. Huawei's U.S. research arm builds separate identity. Reuters (June 24, 2019).

h U.S. Department of Treasury's Office of Foreign Asset Control Sanctions Programs; http://www.treasury.gov/

DOI:10.1145/3352690

Vinton G. Cerf

Polyglot!

Google speaks 106 languages—or at least can understand queries in written form if not also oral form. When I watch someone interacting verbally with Google Assistant in languages other than English (my native tongue), I realize Google's language ability vastly exceeds my own. I have a modest ability to speak and understand German. I know a few phrases in Russian and French. But it suddenly strikes me that Google is usefully dealing with over 100 languages in written and oral form. Assistant is responding to queries by recognizing speech input, searching the Web, and voicing the answers in multiple languages. Google Lens is translating text seen in photos into the viewer's preferred language. Google Translate is converting text and speech in one language into another with increasing quality. The quality varies, of course, depending on the volume of training material available to configure deep neural machine learning networks, but the fact that it works at all across so many languages is nothing short of astonishing and even daunting.

It is examples like these that reinforce my impression machine learning has taken over the computer science world as the tool of choice for a great many applications. As I write this, I am in the middle of judging the Google Science Fair where a good many of the projects on display have their roots in machine learning and recognition or identification of various signals. One project is detecting and amplifying the presence of DNA in river waters to identify species found in or near the river. Another is trying to detect plant diseases by recognizing images of various forms of leaf blight. Another is recognizing sign language by sensing the muscles of the arm as they direct finger movement. Another is trying to sense whether a person is talking, singing, drinking, coughing, or choking by using a sensor taped to the throat; think about its potential uses in remote patient monitoring in an intensive-care facility. Another application is the analysis of heart sounds to detect anomalous valve conditions. A recurrent theme throughout the fair is a desire to apply technology to improving living conditions and, more generally, people's lives.

Given the increasing availability of platforms for implementing machine learning algorithms, it is perhaps not surprising there is rapid exploration of this method for local data analysis and filtering. The term "edge computing" is creeping into the vocabulary with expectations, for example, that machine learning algorithms can be built into mobiles or local processors. I recently installed a device that clamps onto the electrical mains of my house that tries to recognize the signatures of various electricity-consuming devices so as to develop a profile of energy use. As it recognizes new devices, I get excited email messages titled "I've found a new device!" like a school child coming home with a story of discovery from science class. There is something charming and refreshing about this behavior (if I can anthropomorphize a little).

The potential for real-time translation between spoken languages is becoming a feasible reality.

Where is all this taking us? For one thing, the potential for real-time translation between spoken languages is becoming a feasible reality. Think of the Star Trek universal translator and its arrival three centuries ahead of time! Unsupervised learning is taking us closer to discovery as the algorithms discover patterns we might, ourselves, not detect. The recent results of Deep Mind's AlphaZero machine learning system showing it discovering chess moves and tactics never before seen in human play hints at the possibility of new discoveries in other fields. We are collectively exploring vast territories of data like an army of digital Lewises and Clarks. What will be important is deeper understanding of what works and what doesn't and why. When these methods fail there can be catastrophic consequences, so getting this right is a challenge worth meeting.

I am reminded of a grook on problems by Piet Hein:[a]

"Problems worthy of attack
Prove their worth by hitting back!" C

a https://en.wikipedia.org/wiki/Piet_Hein_(scientist)

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

DOI:10.1145/3352489

Moshe Y. Vardi

The Long Game of Research

THE INSTITUTE FOR THE FUTURE (IFTF) in Palo Alto, CA, is a U.S.–based think tank. It was established in 1968 as a spin-off from the RAND Corporation to help organizations plan for the long-term future. Roy Amara, who passed away in 2007, was IFTF's president from 1971 until 1990. Amara is best known for coining Amara's Law on the effect of technology: "We tend to overestimate the effect of a technology in the short run and underestimate the effect in the long run." This law is best illustrated by the Gartner Hype Cycle,[a] characterized by the "peak of inflated expectations," followed by the "trough of disillusionment," then the "slope of enlightenment," and, finally, the "plateau of productivity."

I was reminded of Amara's Law when I heard that the 2018 Turing Award was awarded to Yoshua Bengio, Geoffrey Hinton, and Yann LeCun for "conceptual and engineering breakthroughs that have made deep neural networks a critical component of computing." This decision was hardly surprising. After all, it is difficult to think of any other computing technology that has such a dramatic appearance and impact over the past decade. Quoting the Turing Award announcement: "In recent years, deep-learning methods have been responsible for astonishing breakthroughs in computer vision, speech recognition, natural language processing, and robotics—among other applications."

But it is worthwhile to reflect on the long history of neural nets in order to put this contribution in its proper historical context. In 1943, Warren McCulloch, a neurophysiologist, and a young mathematician, Walter Pitts, wrote a paper on how brain neurons might work. They modeled a simple neural network with electrical circuits. Frank Rosenblatt, a neurobiologist of Cornell, invented the *Perceptron*, a single-layer neural net, in 1958. *The New York Times* reported the perceptron to be "the embryo of an electronic computer that [the Navy] expects will be able to walk, talk, see, write, reproduce itself and be conscious of its existence." Unfortunately, the perceptron is quite limited and was proven as such in Marvin Minsky and Seymour Papert's 1969 book, *Perceptrons*. The peak of hype was then followed by the trough of disillusionment. This so-called "First AI Winter" manifested, among other things, in the declining research funding for artificial intelligence, and lasted until the early 1980s.

In 1982, John Hopfield of Caltech presented a paper with a focus not on modeling brains but on creating useful devices. With mathematical clarity, he showed how such networks could work and what they could do. Around the same time, a *U.S.-Japan Joint Conference on Cooperative/Competitive Neural Networks* was held in Kyoto, Japan. Japan subsequently announced its Fifth Generation effort. U.S. periodicals picked up that story, generating a worry that the U.S. could be left behind. Soon funding was flowing once again. The Annual Conference on Neural Information Processing Systems was launched in 1987. Yet the new peak of hype was again followed by a trough of disillusionment. Quoting again the Turing Award announcement: "By the early 2000s, LeCun, Hinton, and Bengio were among a small group who remained committed to this approach." In fact, their efforts to rekindle the AI community's interest in neural networks were initially met with skepticism. This disillusionment led to the "Second AI Winter," which lasted well into the 1990s.

It was only at the start of this decade that the combination of improved algorithms, improved hardware (GPUs), and very large datasets (ImageNet has more than 14 million labeled images) led to an impressive breakthrough, and it became obvious that deep (many layered) neural nets had significant advantages for machine vision, in terms of efficiency and speed. The ideas of Hinton and his colleagues resulted in major technological advances, and their methodology is now the dominant paradigm in the field, leading to being awarded the 2018 Turing Award.

The moral of this tale is that research is a long game; patience and endurance are necessary components. Yet I remember a research-evaluation meeting in an industrial-research lab in the early 1990s in which someone's seminal work on data mining was not being appreciated, because "he has been doing it for two years now and it is not clear that it is going anywhere." I share the concerns of Abraham Flexner, founder of the Institute for Advanced Study in Princeton; in *The Usefulness of Useless Knowledge*,[b] published 1939, Flexner explores the dangerous tendency to forgo pure curiosity in favor of alleged pragmatism.

There is no single formula for successful research. Sometimes it makes sense to focus short-term on an immediate problem, but, quite often, dramatic breakthroughs are obtained by viewing research as a long game.

Follow me on Facebook and Twitter. **C**

a http://en.wikipedia.org/wiki/Hype_cycle

b https://library.ias.edu/files/UsefulnessHarpers.pdf

Moshe Y. Vardi (vardi@cs.rice.edu) is the Karen Ostrum George Distinguished Service Professor in Computational Engineering and Director of the Ken Kennedy Institute for Information Technology at Rice University, Houston, TX, USA. He is the former Editor-in-Chief of *Communications*.

DOI:10.1145/3349701

On Being 'Random Enough'

THE CONCEPT OF randomness is easy to grasp on an intuitive level but challenging to characterize in rigorous mathematical terms. In "Algorithmic Randomness" (May 2019), Rod Downey and Denis R. Hirschfeldt present a comprehensive discussion of this issue, incorporating the distinct perspectives of "statisticians, coders, and gamblers."

Randomness is also a concern to "modelers" who depend on simulation models driven by random number generators or analytic models built using probabilistic assumptions. In such cases, the underlying mathematical model is often an ergodic stochastic process, and the issue is whether the output of the simulator's random number generator or the observed behavior of the real-world system being modeled is "random enough" to establish confidence in the model's predictions.

In a sense, this highly pragmatic perspective represents a less restrictive approach to the issue of randomness: if any of the strong criteria described by the authors are satisfied, the output of the simulator's random number generator or the observed behavior of the system being modeled should be sufficiently random to establish confidence in a model's predictions. On the other hand, behavior that fails to satisfy these strong criteria may still yield accurate predictions, provided other—less restrictive—assumptions are satisfied.[1] In many cases, these less restrictive assumptions are both simple and intuitively plausible. Their existence explains why many probabilistic models work well in practice even though the rigorous mathematical assumptions these models appear to depend on are unlikely to be satisfied exactly.

Reference

1. Denning, P.J. Rethinking randomness: An interview with Jeff Buzen. *ACM Ubiquity*, Aug. 2016; https://ubiquity.acm.org/article.cfm?id=298632

Jeff Buzen, Nashua, NH, USA

The Human Side of Computing

I am writing to address the continuing issue of women's underrepresentation in computer science and, in particular, to Gloria Townsend's impassioned and well-stated views in "Bringing More Women, Immigrants, to Computer Science" (July 2019) on ways to address this concern. I applaud her efforts and obvious success.

However, I would like to call attention to what might be—in my opinion having taught the subject for over 40 years—one serious cause of this gender disparity: Computer science is far more focused on the technological and informatics aspects of the field rather than the humanistic. In other words, people are spending more time making themselves meaningful to a piece of machinery (sorry, folks, that's all a computer or smartphone is), deciphering how the software and hardware were designed and implemented.

There is nothing particularly socially positive about this effort. All of us have shared the frustration of trying to speak to a person rather than an automated system whose best feature is to tell you a real person will call you back. (The company would argue the best feature of this system is it saves them money on customer service reps.) Indeed, phishing, spamming, spoofing, robocalling, hacking, and cyberwarfare have added to the negative connotations of the discipline. These undesirable elements steer some socially minded students to choose areas of study that showcase and directly contribute to the upside of the species.

In short, modern computing leaves much to be desired as a human-centered endeavor. Therefore, it is only natural that computer science has become less attractive to people who are disposed (by nature, environment, upbringing, experience, or training) to being socially sensitive and people-oriented. I will leave it open for discussion if this characterization applies (statistically) more to women than men, but certainly the community would do well to emphasize the social good computer science is capable of achieving. We must sell the discipline on its immense social value to humanity.

Steven Minsker, Little Rock, AR, USA

Tight Squeeze

In "Extract, Shoehorn, and Load" by Pat Helland (July 2019), I learned a lot about how metadata is used to ensure the most important pieces of data are translated properly, especially when reducing the size of the data being loaded. I think the analogy to shoes for the way we transform data is accurate and interesting; I never would have thought of it like that. Helland goes into detail about how painful the process of fitting data can be but doesn't talk about how it could be improved. I think it would be interesting to hear his ideas on how to make the process less difficult.

Mitch Hudson, Cedar Rapids, Iowa, USA

Author responds:

There are a couple of points to remember. First, there's a difference between the described input and the prescribed output. Second, as I discussed in an earlier article, "If You Have Too Much Data Then Good Enough Is Good Enough" (June 2011), *data transformation can be like a meat grinder. When you make a hamburger, it tastes good, but you can't go backward to the input steak. That's OK. The hamburger's tasty.*

Pat Helland

Out of Focus

In the article, "The Edge of Computational Photography" (July 2019) Keith Kirkpatrick writes, "… has its subject in focus, and the background out of focus, known as bouquet."

The correct word for this effect is "bokeh." Ask any professional photographer!

Ann Ford Tyson, Tallahassee, FL, USA

Communications welcomes your opinion. To submit a Letter to the Editor, please limit your contribution 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:10.1145/3344262 http://cacm.acm.org/blogs/blog-cacm

Why Programmers Should Curb Their Enthusiasm, and Thinking about Computational Thinking

Yegor Bugayenko ponders the dangers of "hazardous enthusiasm," while Mark Guzdial considers whether the need to teach computational thinking can be "designed away."

Yegor Bugayenko
Hazardous Enthusiasm and How Eagerness Can Kill A Project
http://bit.ly/2LHzuLq
June 27, 2019

Programmers are constantly contributing to my open source projects (all of my projects are open source, FYI). Some are volunteering their time, others are paid through Zerocracy. While I have worked with a lot of great developers over the years, I have also come across a number of people afflicted with what I call "hazardous enthusiasm."

These people have energy and often the skills, but are overzealous and don't know how to break down their changes and deliver them incrementally. People afflicted with hazardous enthusiasm frequently want to tear down and rebuild the entire architecture or implement some other huge changes, often simply for the sake of doing so. Let's take a closer look at what I mean.

Let's say a new developer joins the team. At first, he checks all the boxes. He knows his code, he's got energy, he's easy to communicate with, and he's putting in time, submitting new tickets and offering useful suggestions. During those first few days, he seems like a gift from the heavens.

As he learns more about the project, the hazardous enthusiasm starts to creep in. Instead of tickets with helpful suggestions, he hits me up on Telegram with a bold claim: the architecture is a complete and utter mess, and I've got just a matter of weeks before the project will implode.

I counter with a polite reassurance that I understand, but before even hearing me out, he's already suggesting that we re-do everything from scratch. At the very least, he suggests we trash a collection of objects, and replace them with a singleton and a particular ORM library. Of course, he's been using these for months, and they're amazing and as soon as I see everything in action I'm going to be floored and, and, and …

Now at this stage, there's a lot I will probably want to say. I could remind him that I am an architect myself, and that I have a long string of successes under my belt. I might point out that we've been working on this project for some time and that so far development is progressing at a comfortable pace.

Often, however, I say very little and instead ask him to submit a ticket. I offer an assurance: I'll review his suggestions as soon as possible. And I casually remind him that I am an architect, and in fact the architect for this project. In an ideal world, he'd accept that and follow up some incremental changes. More often, he claims that he'll show me how it's supposed to be done.

A few days later, he hits me up with a huge pull request. There are tons of changes, and some of them actually look quite interesting. The problem is, a lot of the suggestions are all but antithetical to the principles I've embedded into the existing architecture. I know he's put a lot of time into his project, but I have to reject the pull request anyway.

Can you guess what happens next? The developer, once a godsend, simply ups and disappears. You see, I'm the bad guy here. I am evil and anti-innovation and closed-minded. How dare I not scrap an entire project and start over!? I've been through all of the above time and time again.

The sad thing is, that developer probably could have made a lot of useful contributions. Sometimes we come across incompetent developers, but a lot of times they're actually great from the technical perspective; what they're lacking is an ability to microtask.

Developers jumping onto new projects need to know how to break down changes into small, digestible chunks and then deliver them incrementally. Instead of pushing out one huge chunk of changes or trying to completely upend the entire project, they need to set their sights lower. As an experienced and successful architect, I'm not going to allow someone to completely implode a project in their first week.

Maybe I'm evil. More likely, the developer has been struck with a case of fatal enthusiasm. Although they want to do the right thing, they are way too eager and overly zealous. Every fix has to be implemented in one pull request and there's no time to wait. Any incremental improvements simply won't be acceptable. Remember, in their view, time is running out and the project is only weeks from failing anyway.

So why don't I just step aside and let them fix the code the way they want? Maybe they're simply a better architect then me. But here's the thing: being a successful architect requires microtasking. As an architect, you have to manage changes, and you have to implement them gradually. This is a basic necessity in a dynamic, collaborative work environment.

The moment a developer comes to me and tries to upend the entire project just a few days in, I already know they are going to struggle with incremental change. That means they're going to struggle in the architect's seat, so I can't exactly hand over the keys to the whole venture.

So no, you are not being evil or closed-minded when you reject hazardous enthusiasm. You are being prudent, wise, or whatever you want to call it. Most importantly, you're being a good architect.

Mark Guzdial
A Design Perspective on Computational Thinking

http://bit.ly/2JkL3q2
June 9, 2019

Computational thinking was popularized in a March 2006 column in *Communications* by Jeannette Wing. In 2010, she published a more concise definition (see her article about the evolution of these definitions at http://bit.ly/2Xwr1Nr):

Computational thinking is the thought processes involved in formulating problems and their solutions so that the solutions are represented in a form that can be effectively carried out by an information-processing agent (Cuny, Snyder, and Wing, 2010).

I have been thinking a lot about this definition (see the BLOG@CACM from last September at http://bit.ly/2S437aS, and my April blog at http://bit.ly/2YBljuV). This is a definition most people can agree with. The problem is when you use it to define curriculum. What does it mean to represent a problem in a form that can be effectively solved by a computer? What do we teach to give students that ability?

Computers are *designed*. The problem form *changes*. We can make computers *easier* to use.

Human-computer interface designers and programming language designers are all about making it *easier* to represent problems in a computable form. A good user interface hides the complexity of computation. Building a spreadsheet is much easier than doing the same calculations by hand or writing a program.

I have been digging deeper into the literature on designing domain-specific programming languages. The empirical research is pretty strong. Domain-specific programming languages lead to greater accuracy and efficiency than use of general-purpose languages on the same tasks (as an example, see http://bit.ly/2NHhFPh). We are learning to make programming languages that are easy to learn and use. Sarah Chasins and colleagues created a language for a specific task (Web scraping) that users could learn and use *faster* than existing users of Selenium could solve the same task (see the blog post at http://bit.ly/2XPd9Sx).

So, what should we teach in a class on computational thinking, to enable students to represent problems in a form that the computer can use? What are the skills and knowledge they will *need*?

- Maybe iteration? Bootstrap: Algebra (http://bit.ly/2YMinvK) showed that students can learn to build video games *and* learn algebraic problem-solving, without ever having to code repetition into their programs.
- Booleans? Most students using Scratch don't use "and," "or," or "not" at all (see the paper at http://bit.ly/2L8ORwL). Millions of students solve problems on a computer that they find personally motivating, and they do not seem to need Booleans.

Our empirical evidence suggests even expert programmers really learn to program within a given domain. When expert programmers switch domains, they do no better than a novice (see the post at xhttp://bit.ly/2NEZidz). Expertise in programming is domain-specific. We can teach students to represent problems in a form the computer could solve in a single domain, but to teach them how to solve in multiple domains is a big-time investment. Our evidence suggests students graduating with a four-year undergraduate degree don't have that ability.

Solving problems with a computer requires skills and knowledge different from solving them without a computer. That's computational thinking. We will never make the computer completely disappear. The interface between humans and computers will always have a mismatch, and the human will likely have to adapt to the computer to cover that mismatch. But the gap is getting smaller all the time. In the end, maybe there's not really that much to teach under this definition of computational thinking. Maybe we can just design away the need for computational thinking.

Comments:

I wonder how well Khan Academy's approach to teaching computational thinking works, since it seems to be more interactive and can be connected to other skills (if there are courses for them): https://www.khanacademy.org/computing

—Rudolf Olah

Yegor Bugayenko is founder and CEO of software engineering and management platform Zerocracy. **Mark Guzdial** is a professor of electrical engineering and computer science, and engineering education research, in the College of Engineering, and professor of information in the School of Information of the University of Michigan.

ACM SIGUCCS 2019 Annual Conference

November 3-6, 2019 | New Orleans, LA

Let the good talks roll!

Registration is Open for SIGUCCS 2019!

Register now and urge your colleagues in higher education IT support to do the same! Come join us in New Orleans this November along with our plenary speakers, Audrey Watters and Jaime Casap!

Audrey Watters
creator of the Hack Education website, and author of the Monsters of Education Technology series

Jaime Casap
Education Evangelist for Google

We're ready to let the good talks roll in the following topic tracks:

- Leadership & Professional Development
- Communication, Training, and Documentation
- Instructional Technology & Design
- Infrastructure, Security, and Operations
- Service Desk and Service Management

Plus four pre-conference seminars and the ever-popular lightning talks and posters!

Learn more about the conference:

http://bit.ly/SIGUCCSRegister

ACM SIGUCCS is the Special Interest Group on University and College Computing Services

N news

Science | DOI:10.1145/3344289 Samuel Greengard

An Inability to Reproduce

Big data and modern analytics offer enormous possibilities for research, provided scientists can produce consistent results.

IMAGE BY PRITSANA

Science has always hinged on the idea that researchers must be able to prove and reproduce the results of their research. Simply put, that is what makes science…science. Yet in recent years, as computing power has increased, the cloud has taken shape, and data sets have grown, a problem has appeared: it has becoming increasingly difficult to generate the same results consistently—even when researchers include the same dataset.

"One basic requirement of scientific results is reproducibility: shake an apple tree, and apples will fall downwards each and every time," observes Kai Zhang, an associate professor in the department of statistics and operations research at The University of North Carolina, Chapel Hill. "The problem today is that in many cases, researchers cannot replicate existing findings in the literature and they cannot produce the same conclusions. This is undermining the credibility of scientists and science. It is producing a crisis."

The problem is so widespread that it is now attracting attention at conferences and in academic papers, and even is garnering attention in the mainstream press. While a number of factors contribute to the problem—including experimental errors, publication bias, the improper use of statistical methods, and subpar machine learning techniques—the common denominator is that researchers are finding patterns in data that have no relationship to the real world. As Zhang puts it, "The chance of picking up spurious signals is higher as the nature of data and data analysis changes."

At a time when anti-science sentiment is growing and junk science is flourishing, the repercussions are potentially enormous. If results cannot be trusted, then the entire nature of research and science comes into question, experts say. What is more, all of this is taking place at a time when machine learning is emerging at the forefront of research. A lack of certainty about the validity of results could also lead people to question the value of machine learning and artificial intelligence.

Methods Matter

A simple but disturbing fact is at the center of this problem. Researchers are increasingly starting with no hypothesis and then searching—some might say grasping—for meaningful correlations in data. If the data universe is large enough—and this is frequently the case—there are reasonably good odds that by sheer chance, a valid p-value will appear. Consider: if a person tosses a coin eight times and it lands on heads every time, this is noteworthy; however, if a person tosses a coin 8,000 times and, at some point, the coin lands on heads eight consecutive times, what might appear to be a significant discovery is merely a random event.

The idea that scientific outcomes may be inaccurate or useless is not new. In 2005, John Ioannidis, a professor of health research and policy at Stanford University, published an academic paper titled *Why Most Published Findings Are False*, in the journal *PLOS Medicine*. It put the topic of reproducibility of results on the radar of the scientific community. Ioannidis took direct aim at methodologies, study design flaws, and biases. "Simulations show that for most study designs and settings, it is more likely for a research claim to be false than true," he wrote in that paper.

Others took notice. In 2011, Glenn Begley, then head of the oncology division at biopharmaceutical firm Amgen, decided to see if he could reproduce results for 53 foundational papers in oncology that appeared between 2001 and 2011. In the end, he found he could replicate results for only six papers, despite using datasets identical to the originals. That same year, a study by German pharmaceutical firm Bayer found only 25% of studies were reproducible.

"Simulations show that for most study designs and settings, it is more likely for a research claim to be false than true."

This is a topic Ioannidis and others have continued to examine, particularly as the pressure to produce useful studies grows. Says Ioannidis, "Today, we have opportunities to collect and analyze massive amounts of data. Along with this, we have a larger degree of freedom about how we collect data, how we assemble it, and how we interpret it." The challenge, then, is to design a methodology around a hypothesis, and then test it with the available data, or use other valid statistical methods when the number of potential hypotheses tested is extremely large. The need for proper methodologies is magnified in an era where data is easily collected and widely available.

Ioannidis expresses concern about the trend toward an "exploration and pattern-recognition approach" that requires little or no planning—and often uses little or no validation of results. Increasingly, he says, researchers resort to the backwards method of using machine learning to identify a hypothesis, rather than starting with one. The data contains hidden patterns, but it's the coin landing on heads eight times in a row out of 8,000 flips result, rather than landing on heads eight out of eight times. "The approach often reflects the mentality that 'something interesting must be there with all the riches of huge datasets'," Ioannidis explains.

Not surprisingly, machine learning can amplify errors and distortions. Inconsistent training methods and poorly designed statistical frameworks lead to patterns and correlations that have no validity or link to causality in the real world. An emerging problem is a lack of understanding about how to use machine learning tools correctly. In fact, a growing number of commercial applications—particularly those designed for the business world—put enormous analytics and machine learning capabilities in the hands of non-data scientists.

Although the reproducibility problem spans virtually every scientific discipline, it is particularly problematic in medicine, where results are frequently unreproducible and the repercussions are greater. Experts say the research community must address the challenge and find fixes because this not only erodes public confidence, it wastes time, money, and valuable resources, all while generating greater confusion about which drugs, therapies, and procedures actually work. Says Ioannidis, "There are potential repercussions—and they can be quite devastating—if doctors make wrong choices based on inaccurate data or study results."

Rethinking Results

Brian Nosek, co-founder and executive director of the non-profit Center for Open Science in Charlottesville, VA, says that if there is a crisis, the current situation represents a "crisis of confidence." Greater degrees of freedom along with motivated reasoning can lead researchers unintentionally down paths that produce less-than-credible findings.

Nosek says it is necessary to reexamine the way researchers approach studies at the most basic level. Among other things, this means emphasizing reproducibility as a key requirement for publication, openly sharing data and code so that methodologies and results can be validated by others in the research community, and promoting transparency about funding and affiliations. In pursuit of this goal, the Open Science Framework (OSF) now offers an online repository where researchers can register studies and allow others to examine the supporting data, materials, and code after the research is complete.

A number of other factors also are crucial to boosting the accuracy and validity of findings. Glenn Begley has observed that six key questions lie at the center of sound research and reproducibility:

- *Were studies blinded?*
- *Were all results shown?*
- *Were experiments repeated?*
- *Were positive and negative controls shown?*
- *Were reagents validated?*
- *Were the statistical tests appropriate?*

By boosting due diligence upfront, Begley argues, it is possible to ensure a much higher level of veracity and validity to research results. The same techniques also apply to analytics and machine learning in business and industry, where users often lack the scientific grounding to ensure the methods they use are sound.

In the scientific community, greater scrutiny can also take the form of more vigorous peer reviews and greater oversight from journals. In some cases, researchers are publishing results that haven't been reviewed at all; they essentially are rubber-stamping their own work. This has contributed to an increased number of retractions and corrections in journals. The *Journal of Medical Ethics*, for example, documented a 10-fold increase in retractions of scientific papers in the PubMed database between 2000 and 2009 alone.

More rigorous statistical methodologies, as well as better use of machine learning, are also critical. As a result, researchers are studying ways to improve analysis. For example, instead of conducting exploratory data analysis on an entire data set, researchers might use data splitting—essentially, separating a training dataset and test dataset and keeping the test dataset hidden until the end, once the results are generalizable.

Another approach involves taking an original training dataset and randomizing it in a way that mimics future datasets by adding random noise repeatedly. If researchers can aggregate all the results and the discovery remains stable (meaning it appears across many different randomized datasets), then it's likely to be reproducible.

Forward Thinking

Although the inability to reproduce scientific results has grown in recent years, observers say most researchers strive for accurate findings and the problem is largely solvable. Enormous datasets and the widespread use of machine learning are relatively new additions to mainstream science, and it is simply a matter of time before more stringent methodologies emerge, they say.

"We are in the midst of a reformation. The research community is identifying challenges to reproducibility and implementing a variety of solutions to improve. It is an exciting time, not a worrying one," Nosek argues.

Zhang also says there's no reason to push the panic button; scientific methods are messy, difficult, and iterative. "We need to embrace changes. We need to be more selective and careful about avoiding mistakes that lead to irreproducible results and invalid conclusions. Right now, this crisis represents enormous opportunities for statisticians, data scientists, computer scientists, and others to develop a more robust framework for research."

Adds Ioannidis, "I'm optimistic that we will find ways to solve the problem of irreproducibility. We will learn how to use today's tools more effectively, and come up with better methodologies. But it's something we must confront and address." C

Further Reading

Ioannidis, J.P.A.
Why Most Published Research Findings Are False, *PLOS Medicine*, Aug. 30, 2005. https://doi.org/10.1371/journal.pmed.0020124

Berk, R., Brown, L., Buja, A., Zhang, K., and Zhao, L.
Valid Post-Selection Inference, *The Annals of Statistics*, 2013, Vol. 41, No. 2, 802-837. https://projecteuclid.org/euclid.aos/1369836961

Aschwanden, C.
Science Isn't Broken: It's Just a Hell of a Lot Harder Than We Give It Credit For, *FiveThirtyEight*. Aug. 19, 2015. https://fivethirtyeight.com/features/science-isnt-broken/#part1

Halsey, L.G., Curran-Everett, D., Vowler, S.L., and Drummond, G.B.
The Fickle P Value Generates Irreproducible Results. *Nature Methods*, March 2015, Vol. 12 No. 3. pp. 179. https://www.nature.com/articles/nmeth.3288.pdf?origin=ppub

Samuel Greengard is an author and journalist based in West Linn, OR, USA.

ACM Member News

USING EXPERTISE TO EXPAND ACCESS TO THE INTERNET

"Growing up, the 1980s was a computer bubble, and it was the era of the personal computer," recalls Elizabeth Belding, a professor in the department of computer science at the University of California, Santa Barbara (UCSB). "I liked the cutting-edge aspect of technology, which brought me to computer science," she adds.

Belding earned her master's degree and doctorate in electrical and computer engineering from UCSB. On completion of her Ph.D., she joined the faculty of the computer science department at UCSB, where she has been working ever since.

The focus of Belding's research is on mobile and wireless networking, including network performance analysis, and information and communication technologies for development (ICTD).

"I have always been in mobile wireless networking," Belding explains. "I started in protocol development, and then got into network performance analysis."

She recalls that about a decade ago, "I wanted to do something with social impact, and applied my wireless networking expertise to bring Internet access to more people and communities worldwide." Belding adds her work is now largely concentrated on Native American groups within the U.S.

Some of Belding's interests have moved beyond networking. All of her ICTD work falls under the category of computing for social good, or computer science that has a high social impact. Other projects on which she is currently collaborating include analyzing hate speech, and also gender-based violence online and in social media.

"I like what I am doing, and will continue to work on socially impactful projects," Belding says.

—John Delaney

Technology | DOI:10.1145/3344293

Gary Anthes

Augmented Reality Gets Real

Formidable optical challenges are yielding to intensive research, development.

"THIS IS OUR most desperate hour," said the flickering blue image. "Help me, Obi-Wan Kenobi. You're my only hope." In 1977, in a classic moment in cinematic history, the *Star Wars* movie epic gave the public a preview of augmented reality (AR).

But it was only a preview. The three-dimensional (3D) hologram of Princess Leia standing on a real table had been simulated by special effects artists at Lucasfilm, but it was easy to imagine a day when consumers could bring actual 3D virtual images seamlessly into their physical environments and interact with them in real time. That is now possible in a variety of consumer, medical, and industrial applications, generally through the use of head-mounted displays (HMDs).

AR, sometimes loosely called "mixed reality," combines virtual reality (VR) with the physical world. A recent application, offered by Schell Games, uses technology from Disney and Lenovo to bring Darth Vader into your living room—your *actual* living room. In the game Jedi Challenges, users with a smartphone-enabled headset, a lightsaber controller, and a tracking beacon can engage the life-sized movie villain in a lightsaber battle.

Battling a virtual Kylo Ren in the augmented reality game Star Wars: Jedi Challenges.

IMAGE COURTESY OF LENOVO.COM

The AR game has received enthusiastic reviews for its realism, yet no one would for an instant think the fallen Jedi Knight is actually standing between the sofa and the coffee table. Darth Vader has the same unreal blue hue as the Princess Leia avatar; he is semi-transparent, pixelated, subject to ghosting, and lacking in detail.

Yet the market for AR headsets is potentially huge, and the optical challenges in AR are the subject of intense research by companies and universities around the world. MarketWatch, published by Dow Jones, forecasts the AR market will grow at a compound annual rate of 75% to reach $50 billion by 2024, just five years from now. Most analysts look for the greatest growth in retail, automotive, and medical applications, although some predict eventually consumer AR devices will replace all conventional displays on laptops and smartphones.

The optical challenges when combining VR and AR are complex, and they have yielded ground only grudgingly. Depending on the application and the technology for it, images seen by users are often primitive: blurred, low in resolution, slow to refresh, subject to a narrow field of view, or generally unrealistic in look and behavior. Moreover, user gear can be expensive, aesthetically unpleasing, and uncomfortable to use, especially for a prolonged period of time.

There are a myriad of technical approaches to each of these problems, but they tend to solve one problem by creating others. A solution that is adequate for a living room game may be a complete failure in an operating room or construction site.

Problems, Trade-Offs

Martin Banks, a professor of optometry and vision science at the University of California, Berkeley, and director of the Center for Innovation in Vision and Optics, says the problems with the optical components in AR systems fall into two broad categories: how good the image is as seen by the eye, and how well addressed is the "vergence-accommodation conflict."

The optical quality issue presents classic trade-offs. Diffractive lenses, often used in AR systems because they are small and light, suffer from quality issues—such as blur and color fringing—compared with larger, heavier refractive lenses. "To get good image quality, you generally need a refractive lens, and there goes your form-factor improvement," Banks says.

Optical quality also depends on another difficult trade-off: the quest for a realistically wide field of view. "You want a large field of view, but once you spread out a fixed number of pixels, you can see them," Banks says. Similarly, although the human eye can detect very fine detail (up to 50 alternating light and dark stripes per degree of vision), HMDs today typically can deliver just seven to 12 lines per degree, a limitation of the lens, the display, and the computer power available to process images.

The other major problem, the vergence-accommodation conflict, is even more difficult. The eyes converge (turn inward) to see objects that are near, but diverge to see far-away objects. That alignment of the two eyes is known as "vergence," and when it fails, the viewer sees double images. A separate system, called "accommodation," changes the shape of the lenses in the eyes so a person sees images in sharp focus. "The brain has a circuit that links these two systems so that you nearly always converge and accommodate to the same distance," Banks says.

However, AR systems often present images for which those distances are not the same. The efforts of the eyes and brain to deal with that conflict can cause eye discomfort, and even nausea. Some users can't cope at all, and just see doubled or blurry images. The conflict is the subject of intense research at companies worldwide, at Berkeley and Stanford in the U.S., and at universities in China, Korea, Canada, England, and Holland, Banks says.

There are two ways currently to address the vergence-accommodation problem, and they are equally applicable to virtual reality and augmented reality systems. In "varifocal" devices, a focus-adjustable lens sits in front of each eye and, enabled by an eye-tracker, is able to estimate where the user is looking and then adjust the focus of the lens to make vergence and accommodation consistent with each other. Switzerland's Optotune is a leading maker of such "focus-tunable lenses."

The need to make wearable computers and power supplies small and lightweight constrains the amount of compute power available for high-quality, real-time rendering of 3D images.

The varifocal technique works, Banks says, but it adds cost, weight, complexity, and power consumption. A related approach is found in multifocal AR systems. With varifocal lenses, the focal length can change continuously, but in multi-focal, digital circuits chose among two fixed focal planes.

The need to make wearable computers and their power supplies small and lightweight constrains the amount of compute power available for high-quality, real-time rendering of 3D images. A possible solution is a new technique called foveated rendering or gaze-contingent foveation, which uses an eye-tracker to put the best image quality only in front of the eye's fovea, where visual acuity is greatest. "If we can track your eye, not your head, we can tell what you are looking at," says Tom Corbett, an instructor in entertainment technology at Carnegie Mellon University. "Then we can render in high resolution only what you are looking at, and all else we can do at a slower frame rate."

How They Work

Wearable devices for consumer AR typically employ one of two basic designs, each of which has variants. The first, the "curved mirror," uses a semi-reflective, semi-transparent concave mirror placed in front of the eyes and connected to an off-axis projection system. An example is the Lenovo-Disney Jedi Challenges headset. Distortion and ghosting can be a problem with this method, requiring optical or electronic correction that adds weight and cost to the system, while reducing resolution.

The second basic approach, "waveguides," uses diffractive, reflective, holographic, or polarizing optics to guide uni-directional waves of light from a side-mounted source to create an image in front of the eyes. Waveguide technology is used by a number of companies, including Microsoft in its HoloLens 2 HMD, Magic Leap in its headset, and by Akonia Holographics (purchased last year by Apple). Waveguide-based HMDs are smaller, lighter, and offer good optical quality, but they are (so far) more expensive, and offer more limited fields of view compared to curved mirrors.

Jesse Schell, CEO of Schell Games and a professor at Carnegie Mellon's Entertainment Technology Center, predicts waveguides and curved mirrors will ultimately be dominated by a third approach which, so far, "no one likes to talk about." Instead of looking at the real world through a piece of transparent glass, users have a headset with built-in cameras that show the user video of the real world blended with digital virtual images. One drawback with that method results from the time lag between when a camera captures an image and when it is presented to the eye, Schell says. This latency, often about 100 milliseconds, can cause nausea as the brain struggles to adjust. Solving the problem will require low-latency cameras driven by greater computer power for faster image refresh.

"With this video pass-through method, you don't have the struggle with all the tricky optics, there is no field of view problem, no brightness problem, and the image quality will get better as low-latency cameras appear," Schell says. "It will become a strikingly good experience, and very affordable."

Schell predicts video pass-through

will capture 80% of the market for industrial and military applications within five years.

Solving the Vergence-Accommodation Problem

Magic Leap uses a multi-focal approach to the vergence/accommodation conflict, in which the system dynamically chooses between two focal planes, whichever minimizes the conflict to a greater degree. That ensures the user does not see a conflict that exceeds a small "mismatch budget," says Michael Klug, vice president for advanced photonics. He says the company has built HMDs that can shrink that budget to an arbitrarily small amount by continuously sweeping across many focal planes. However, Klug says such devices at present are too large, too complex, and use too much computer power.

Banks at UC Berkeley says the "holy grail" solution to the vergence-accommodation conflict is a complicated and experimental technology called "light field." A pixel in most displays today has two dimensions: a vertical position and a horizontal position. A light-field display adds two more (vertical and horizontal direction), so vectors of light are sent to the eye. Banks says some companies today incorrectly use the term "light field" to describe their products, but they don't in fact "create a reasonable approximation to the light field we experience in everyday life."

"If we can track your eye, not your head, we can tell what you are looking at. Then we can render in high resolution only what you are looking at, and all else we can do at a slower frame rate."

Ronald Azuma, Augmented Reality Team Leader at Intel Labs, declined to be interviewed for this article but provided a video of a presentation to an industry group last year in which he said that Intel had shown experimentally that workable light-field displays are possible, but are not yet commercially practical.

Thanks to robust research efforts, Azuma predicts progress will be made on that and on many other fronts in AR. In fact, he predicts consumer AR will become "as ubiquitous and invaluable as smartphones."

In the longer term, Azuma says, "AR could become the dominant platform and interface, moving us away from fixating on display screens." C

Further Reading

Azuma, R.
The Road to Ubiquitous Consumer Augmented Reality Systems, *Human Behavior and Emerging Technologies*, 01 Feb. 2019
https://doi.org/10.1002/hbe2.113

Billinghurst, M., Clark, A., and Lee, G.
A Survey of Augmented Reality, *Foundations and Trends in Human–Computer Interaction*, Vol. 8: No. 2-3, pp. 73-272, 31 Mar. 2015
https://www.nowpublishers.com/article/Details/HCI-049

Koulieris, G., Bui, B., Banks, M.S., and Drettakis, G.
Accommodation and comfort in head-mounted displays, *ACM Transactions on Graphics (SIGGRAPH Conference Proceedings)*, Vol. 36, No. 4, page 11 July 2017
http://s2017.siggraph.org/sites/default/files/firstpages_part_07.pdf

Kramida, G.
Resolving the Vergence-Accommodation Conflict in Head-Mounted Displays, *IEEE Transactions on Visualization and Computer Graphics*, Vol. 22, No. 7, July 1 2016
https://www.cs.umd.edu/sites/default/files/scholarly_papers/Kramidarev.pdf

Gary Anthes is a technology writer and editor based in Arlington, VA, USA.

Milestones

U.S. Election Assistance Committees Include Past ACM President

Michael Yaki, chair of the Board of Advisors to the U.S. Election Assistance Commission, recently announced the creation of several special committees to support the commission's work.

Said Yaki, these moves constitute "the first steps towards ensuring that we deliver on our mission to advise the EAC."

Yaki said the new Election Security committee will take the lead in recommending measures for local governments in preparing for the 2020 election, as "No other issue is of more importance to the American electorate, and decisions are being made now that will have ramifications for the integrity and security of the 2020 vote." He said the Election Security committee will work quickly to develop priorities and recommendations applicable to all election jurisdictions' voting systems.

Leading the Election Security committee as its chair will be computer scientist, ACM U.S. Public Policy Council (USACM) founder, and former ACM president Barbara Simons. Lawrence Norden, deputy director of the Brennan Center's Democracy Program, will serve as the committee's vice-chair. The committee's first members are Shaun Rahmeyer, administrator of the Nevada Office of Cyber Defense Coordination; Sachin Pavithran, public member of the U.S. Access Board Providence, a federal agency that promotes equality for people with disabilities through leadership in accessible design and the development of accessibility guidelines and standards, and Philip Stark, associate dean in the Division of Mathematical and Physical Sciences, and professor of statistics, at the University of California, Berkeley.

The new Research committee will take advantage of expertise on the board to assist in and advise on research efforts by the EAC.

The Research committee will be led by chair Alysoun McLaughlin, who is deputy election director for Montgomery County, Maryland. Serving as vice-chair is Neal Kelley, registrar for voters in Orange County, CA. The initial members of this committee are Linda Niendick, county clerk of Lafayette, Missouri; James Dickson, co-chair of the Voting Rights Task Force of the National Council on Independent Living, and Elliot Berke, managing partner of the Washington, DC-based law firm Berke Farah LLP.

Yaki said more committee announcements will be forthcoming, "as the members of the Board are eager to ensure that the mandate of the EAC and the Help America Voter Act are fulfilled."

Society | DOI:10.1145/3344291 Sarah Underwood

Can You Locate Your Location Data?

Smartphone apps offering location data services may be desirable, but their ability to collect personal data that can be sold to third parties is less attractive.

SHOPPING TRIP, SCHOOL pick-up, medical appointment, first date, divorce court, or something even more personal—whenever you carry a mobile device and wherever you take it, chances are data about your location is being collected.

Smartphone apps that are free to use on the basis that individuals find them desirable and therefore agree the apps can collect their location data seem to be reasonable, but exactly what data is collected, how it is used, and whether it is sold to third parties is a much bigger, darker picture that smartphone users are unlikely to see at first glance, and will only experience when their data is used for purposes far beyond their initial consent.

With a stringent data privacy law expected to be introduced in California in January 2020, increasing litigation on the potential misuse or abuse of personal location data, and rising public awareness that smartphone apps are not necessarily what they seem to be, location data has become a hot, contentious topic.

Frank Yoder, head of marketing at MightySignal, is a specialist in mobile app intelligence and the software development kits (SDKs) that are integrated in apps to perform anything from location data collection to in-app purchases and data monetization. Yoder says there are 43 location tracking SDKs for Apple iOS devices and 39 for Android devices. In total 6,725 apps run these SDKs.

MightySignal does not touch data or track location data per se, instead operating an SDK intelligence platform that continuously monitors apps and which SDKs they are installing or uninstalling, for reasons such as how well the SDK integrates with the app or whether

One's location history often amounts to leaving a digital trail that is easily captured via smartphone apps.

there are better apps in the market using different SDKs. The company's customers are mostly SDK providers tracking their competitors and looking for opportunities as app providers change their SDK technology stacks.

SDK developers include Foursquare, which recently acquired Placed from Snap, parent of Snapchat, as well as Cuebiq, GroundTruth, and Factual. SDK developers also collect data and provide commercial location data services to help clients drive smarter digital products for better marketing and business decisions.

Cuebiq, by way of example, collects anonymous data from mobile devices when users download one of its partner apps and opt in to the app's location services. The company partners on more than 220 mobile apps that include the proprietary Cuebiq SDK. Valentina Marastoni-Bieser, executive vice president of marketing at Cuebiq, says the opt-in process is straightforward and explicit. When a user signs up for one of Cuebiq's partner apps, a prompt appears on their device to opt in to the app's location services. When users opt in, Cuebiq collects their data anonymously; they have the option to opt out at any time.

The resulting data is aggregated and analyzed for high-level, macro visitation trends, meaning the data collected does not contain any personally identifiable information. Brands and advertisers can access insights derived from Cuebiq's data using its artificial intelligence-driven business intelligence platform, Clara, which helps them map and measure offline customer trends, and shape strategic business decisions such as when to do promotions or where to build new locations. Its typical client sectors include quick service (or fast food) restaurants (QSRs), retailers, financial services, and automotive businesses.

Snap collects location data through its Snapchat app. It does not sell the data, but does use it for its own commercial purposes, such as location-based ad targeting and provision of services such as geofilters that allow mobile users to add a location illustration to their 'snaps', and Snap Map, which lets users share their location with friends on a map. This is turned off by default and can be switched on and off at any time. Rather than selling data services, Snapchat allows brands to advertise to its audience through a mixture of in-app advertising formats such as Snap ads and commercials.

Keen on data privacy, Snap's privacy policy states: "When you use our services, we may collect information about your location. With your permission, we may also collect information about your precise location using methods that include GPS, wireless networks, cell towers, Wi-Fi access points, and other sensors, such as gyroscopes, accelerometers, and compasses."

Users must give device-level permission for location data to be collected by the Snapchat app. Prior to Snapchat collecting location data from users, it requires express consent through an in-app location consent pop-up. Users can opt out of location data collection and location sharing at any time, and they can delete most of their stored data at any time in the 'setting' tab of the app. Under GDPR, European Union users also have the 'right to object' to Snapchat's use of their information.

Policies like these are used by most app publishers, but for personal location data to have become such a controversial and touchy topic, something else must be going on under the hood.

A recent study by online security start-up vpnMentor, set up by ex-Google marketer Ariel Hochstadt, reviewed the privacy policies of some of the most popular apps to discover how they really track individuals' every move. In terms of location, the study reports that apps such as Tinder continue to track your location when the app is not in use. Facebook and Instagram not only track your location, but also save your home address and most commonly visited locations.

Anonymity, often proclaimed by location data collectors and providers, can raise similar unforeseen issues for smartphone users. While some applications of location data, such as those in financial services, have no interest in individual profiles and offer broader economic information, data collected by many apps is not truly anonymous. Personal identifiers such as names can be removed from the data, but the data may also include elements tied to individuals, such as a device service number or IP address. If these elements are not eliminated, they can be used to re-create profiles that may not have a name but are not anonymous.

Serge Egelman, research director of the Usable Security & Privacy Group at the International Computer Science Institute (ICSI) and a member of the department of electrical engineering and computer sciences at the University of California, Berkeley, says the biggest issues around personal location data are that consumers do not necessarily have an indication of when their data is being collected, and also have a poor understanding of how that data is used.

Egelman cites reports earlier this year of law enforcement gaining access to Google's mobile location history database, known internally as Sensorvault, using 'geofence' warrants that specify a geographic area and time period. Google gathers information from the database about devices meeting the warrant criteria and initially labels them with anonymous identity numbers. Detectives look at locations and movement patterns to see if any appear relevant to their investigation and, if so, ask Google for names and other sensitive information. Such situations are helpful to law enforcement, but constitute an abusive use of personal data for those in the geofence area who are identified but have done no wrong.

Of course, individuals have to give Google permission to collect their data, which they usually do, even though they don't necessarily know the details of the data that is collected and thatit is kept for an indefinite period. As Egelman says, "You wouldn't expect to be implicated in a crime just by using a Google service; that's scary. How can people control this kind of thing when they don't know what it is?"

Data collected by many apps is not truly anonymous. Personal identifiers can be removed from the data, but the data may include elements tied to individuals.

While both the iOS and Android platforms have permissioning systems that come into play when an app tries to access location data and respond to user decisions on whether they want location data turned on or off, Egelman says the problem is that there is no real context. When the users say 'yes' to location data, they don't know whether this is more desirable for the user or for the app tracking the user. The first time the user clicks the button for data, the data may be desirable, perhaps pointing to shops nearby. Once the button is clicked, however, these platforms use that permission in perpetuity, and that use could be for something that is not desirable to the user.

There is also the issue of apps that collect data with consent from their users, and sell it to third-party advertisers without the consent of users. This problem is manifested in an ongoing legal dispute that started early this year between the Los Angeles city attorney and The Weather Channel app, a subsidiary of IBM. The lawsuit claims the app did not adequately disclose to users how their location information would be used, and calls The Weather Company's practices 'fraudulent and deceptive', saying they violate California's Unfair Competition Law. The lawsuit explains that after downloading the app, users are prompted to allow it to access their location data, but how that data will be shared, or sold, is not noted.

"The permission prompt also fails to reference or link to any other source containing more detailed information about what users' geolocation information will be used for," states the lawsuit. The app's privacy policy does note that data could be used for targeted advertising and might be shared with partners, but the attorney argues that users have no reason to look at the policy, as the prompt does not suggest their data will be used in these ways.

Los Angeles city attorney Michael Feuer told *The New York Times*, "If the price of getting a weather report is going to be the sacrifice of your most personal information about where you spend your time day and night, you sure as heck ought to be told clearly in advance." This is not a one-off problem, nor one with an easy solution. As Egelman asks, "How can you make a decision about how data is used, when it is sold on?"

Egelman's research team has been examining how mobile apps access sensitive data, and recently commercialized a search engine called AppCensus (https://search.appcensus.io/) that looks up the privacy behaviors of free apps (it is still free for consumers). The platform acted as a test bed for research into the behavior of 6,000 free Android children's apps. The team reported that more than half the apps shared details with third-party companies that may have violated the Children's Online Privacy Protection Act (COPPA), which provides digital privacy protections (including for location data) for children under 13.

While location data made its name in advertising, initially by the likes of Facebook, Google, and Twitter, it has since become key to sectors such as financial investment, where firms are looking for new ways to find investment returns that exceed a market index or benchmark. Providers of location data services (or alternative (alt) data, as it is known in this heavily regulated sector) are scrupulous about data privacy, do not use personal data for compliance reasons, and only use alt data to demonstrate trends. They are, however, very knowledgeable about its promises and failures.

Abraham Thomas, chief data officer at Quandl, a provider of insights from alternative data, recalled, "Twenty years ago, Wall Street investment firms sent junior analysts to malls at the weekend to count cars; if there were a lot of cars, the economy was booming. If there weren't, it was in recession."

Technology can now track retail customer activity in near-real-time across the entire country, covering every mall and every store. However, as Thomas points out, to be useful, location data needs to be accurate in every sector. Problems here include the vast number of smartphones in the world, poor GPS signals, and cellular coverage that is limited by thick walls and sends truncated location data that adds bias to the data.

New York City-based Thasos Group also provides what it calls 'actionable information from real-time location data' for investment firms, and publishes an annual Retail REIT (Real Estate Investment Trust) performance update. Thasos co-founder and chief product officer John Collins says the company licenses location data from app aggregators, primarily SDK providers and sometimes brokers.

"This is a market failure based on data asymmetry. Consumers don't have enough information, and those that could provide [it] choose not to."

Thasos products are built on location data collected with user consent and sold on an anonymous basis. The company insists, like Snap, that apps must include secondary consent disclosure, which mean the app will collect location data among other data, use the data for a variety of purposes, monetize the data by selling it on an anonymous basis, and allow users to opt out of the process by turning off location services.

This plays into the requirements of the forthcoming California Consumer Privacy Act (CCPA) that will give residents of California extended rights around their personal data. Specifically, they will be able to:

- know what personal information is being collected about them;
- access that information;
- know if their personal information is disclosed, and if so to whom; and
- know if their personal information is sold, and have the right to opt out of the sale.

The legislation extends beyond existing U.S. data privacy laws, although the rights it bestows are not as prescriptive as those of the European Union's General Data Protection Regulation (GDPR), which took effect in May 2018.

As well as improving data privacy, CCPA raises questions about users having to turn off many of their apps if they don't want to share their location data. Collins suggests few apps, mainly weather and navigation apps, actually require location data to function, and that a number of apps that collect location data have stopped selling the data. According to Collins, "It is better for apps to keep their user base than require secondary disclosure. Where location data is not strictly needed by an app, users can still use its core functionality."

With the implementation of the California privacy law just months away, attorneys are lobbying legislators about a federal law, which is almost certain to follow. It is unclear how the federal government will act, but any legislation is expected to be less onerous in terms of personal consent requirements than the California law.

Meantime, Egelman concludes, "This is a market failure based on data asymmetry. Consumers don't have enough information, and those that could provide the information choose not to. This discrepancy is coming to a head right now. We need more regulation." C

Further Reading

California Consumer Privacy Act
https://leginfo.legislature.ca.gov/faces/billTextClient.xhtml?bill_id=201720180AB375

General Data Protection Regulation
https://eur-lex.europa.eu/legal-content/EN/ALL/?uri=celex%3A32016R0679

Won't Somebody Think of the Children? - Examining COPPA Compliance at Scale, Berkeley Laboratory for Usable and Experimental Security (BLUES), April 25, 2018,
http://bit.ly/2IkZpX0

Who's Watching You?, *vpnMentor*,
https://www.vpnmentor.com/research/whos-watching-you/#/

Google's Sensorvault Is a Boon for Law Enforcement. This Is How It Works, *The New York Times*, April 13, 2019
https://nyti.ms/2MVf9EF

L.A. is suing IBM for illegally gathering and selling user data through its Weather Channel app, *Los Angeles Times*, Jan. 4, 2019
https://lat.ms/2FfGxa2

Thasos 2019 Retail REIT Performance Update, Thasos Group
http://thasosgroup.com/blog/2019-retail-reit-performance-update/

Sarah Underwood is a technology writer based in London, U.K.

DOI:10.1145/3349270

Eric Goldman

▸ James Grimmelmann, Column Editor

Law and Technology

Internet Immunity and the Freedom to Code

A call to preserve the capability of developing the next generation of Internet services.

THE INTERNET'S FREEDOM to code is in jeopardy. In 1996, Congress enacted 47 U.S.C. § 230 ("Section 230"), which says Internet services are not liable for third-party content in many cases. In practice, for over two decades, Section 230 has legally immunized coders' decisions about how to gather, organize, and publish third-party content.

Section 230 has become a political target by all sides, but reforming it will impair coding freedom. In this Law and Technology column, I explain how Section 230 came into existence, the effects it has had, and why technologists should rally behind it to preserve their ability to build the next generation of Internet services.

Section 230's Origins and the Moderator's Dilemma

Two Seminal Cases. Two 1990 court rulings laid the foundation for Section 230. (For more about Section 230's history, see Jeff Kosseff's excellent book, *The Twenty-Six Words That Created the Internet*); see https://bit.ly/2G8ATH7.

In 1991, in *Cubby v. CompuServe*, CompuServe defeated a defamation claim for carrying a third-party publication called *Rumorville*. The court said CompuServe could be liable if it knew or should have known about the defamation. However, CompuServe lacked that knowledge because *Rumorville* uploaded its content directly to CompuServe's servers, without any human pre-screening by CompuServe, and CompuServe had not been told about the defamation.

Despite CompuServe's win, the *Cubby* ruling was not great for other online services that publish third-party content. First, CompuServe passively hosted *Rumorville* and exercised no editorial control over it. While passive hosting might work for some professionally produced content, the rough-and-tumble universe of user-generated content usually requires more active management.

Second, if Cubby had notified CompuServe of the alleged defamation, CompuServe would have had to remove the content to avoid liability. Defamation is easy to allege—and difficult for Internet services to evaluate. Thus, under *Cubby*'s rule, Internet services would have routinely received takedown notices claiming third-party content was defamatory, and the services would have honored the notices regardless of their legitimacy. Indeed, we have seen analogous problems with the Digital Millennium Copyright Act's notice-and-takedown scheme for claiming that users infringed copyright.

A 1995 decision, *Stratton Oakmont v. Prodigy*, delivered even worse news to the Internet industry. Prodigy advertised itself as a "family-friendly" service. It operated popular message boards. A user posted messages that allegedly defamed the plaintiff (the investment bank unfavorably portrayed in the 2013 movie *Wolf of Wall Street*). The court said that Prodigy was the legally responsible publisher of user-submitted posts because Prodigy had removed other user postings from its message boards and touted itself as family-friendly.

The *Stratton Oakmont* decision created a paradox called the "Moderator's Dilemma." According to *Stratton*

Oakmont, moderating user content increased the service's potential legal liability for any harmful content it missed. Accordingly, services had to moderate user-submitted content perfectly or accept liability for any mistaken decisions. Alternatively, it might be legally wiser for Internet services to passively host user content—like CompuServe's passive distribution of Rumorville—than to do any moderation at all. Following *Cubby* and *Stratton Oakmont*, the Internet community was not sure which approach was better.

The Online Pornography Overreaction. In 1995, sensational (and largely overblown) stories reported that children could easily access pornography online. Congress responded to this panic with a new crime that would send online service operators to jail if they allowed children to access pornography.

Two Congressmen, Reps. Cox and Wyden, envisioned a different approach. Instead of banning online pornography, they thought online services would voluntarily curb user-submitted pornography—if the services did not face the Moderator's Dilemma created by the *Stratton Oakmont* decision. Thus, Cox and Wyden proposed shielding online services from liability for third-party content, with the hope that online services would feel legally secure enough to perform content moderation duties that benefit everyone. That proposal became Section 230.

Though the criminal liability provisions and Section 230's immunity were intended as alternatives, Congress combined them into a single law called the Communications Decency Act ("CDA"). In 1997, the U.S. Supreme Court struck down the CDA's criminal provisions, leaving Section 230 in place.

What Section 230 Does

Section 230 gives technologists enormous freedom to design and implement user-generated content services. As a federal appeals court explained in 2016 while ruling in favor of Section 230's immunity: "[the plaintiff's] claims challenge features that are part and parcel of the overall design and operation of the website … Features such as these, which reflect choices about what content can appear on the website and in what form, are editorial choices that fall within the purview of traditional publisher functions."

This legal standard facilitates innovation in several ways.

First, services may freely experiment with new ways of gathering, sorting, and presenting user-generated content. Under different liability rules, those experiments would expose the services to liability for any harmful content they missed, discouraging experimentation and innovation. For example, plaintiffs have argued that user-generated content sites should face liability for different ways they algorithmically promote or excerpt user content. For now, Section 230 forecloses those arguments.

Second, Section 230 helps innovative services launch without having been perfected, so services can error-correct and fine-tune their technology in response to actual usage. For example, new online services can launch without replicating Google's $100M+ investment in filtering technology or hiring

IMAGE BY ROMAN ART

tens of thousands of content reviewers like Facebook has. Without Section 230, online services would need to deploy industrial-grade content controls at launch, which would significantly raise the costs of entry and make it impossible for many innovative services to reach the market at all.

Third, Section 230 permits diverse industry practices to emerge. If the law required mistake-free content moderation, the industry would gravitate toward a single content moderation technological solution that minimized liability. Instead, Section 230 enables services to choose among a virtually infinite number of content moderation techniques—allowing services to optimize their content moderation for their specific audience's needs. Thus, Facebook can tightly restrict hate speech, while Reddit can tolerate subreddits that span a wide ideological spectrum. Similarly, services competing for an identical audience can deploy different solutions that become additional points of competitive differentiation.

Due to Section 230, industry "best practices" for content moderation did not get set in stone during the Internet's earliest days. Instead, best practices for user-generated content continue to iterate, and those iterations potentially deliver ever-increasing social benefits from content moderation.

Furthermore, because of Section 230, lawyers typically do not define product specifications for new user-generated content services; technologists and marketing experts do. It means coders can code without waiting for legal clearance. In a less-favorable legal environment, that will be reversed.

Section 230's Imperiled Future

Section 230 was a bipartisan law, and it garnered significant bipartisan support for its first two decades. That has changed. It has few friends today, and both Democrats and Republicans have publicly targeted Section 230. Congress' gridlock is notorious, but reforming Section 230 has the potential to gather enough bipartisan cooperation to break through that gridlock.

We have already seen one bipartisan incursion into Section 230. In 2018, Congress passed FOSTA, a law that reduced Section 230's immunity for the advertising of commercial sex. Congress was repeatedly warned that FOSTA would not actually solve the problems it targeted, and that it would resurrect the Moderator's Dilemma that Section 230 had negated. Despite those warnings, Congress overwhelmingly passed FOSTA. Worse, its sponsors still consider the law a success, despite the growing evidence that FOSTA has not helped victims and has eliminated or degraded free speech on the Internet.

Often, regulators assume services can just "nerd harder," that is, magically solve an impossible technological task.

FOSTA is part of a growing global trend of imposing criminal liability on online service executives for not adequately restricting harmful user content. Australia recently enacted similar liability, and the U.K. has proposed doing so as well. The risk of criminal liability devastates entrepreneurship because entrepreneurs are willing to risk money, but they will not risk their personal liberty.

Reducing Section 230's immunity invariably would impair the freedom to design and innovate. (The First Amendment might provide some backup protection, but not enough). For example, to reduce online service "bias," some conservative regulators favor "must-carry" obligations that force online services to treat all content equally (despite decades-long conservative opposition to must-carry obligations in other media). Such a rule would be a policy disaster because it would enable spammers, trolls, and miscreants to overwhelm services with their anti-social content and behavior. This policy would also inhibit new technological ways to filter and present content because the law would dictate a single option.

Often, regulators assume that services can just "nerd harder," that is, magically solve an impossible technological task. For example, Europe's Copyright Directive requires online services to ensure no copyright infringing material appears on their services; and Germany's NetzDG law requires online services to remove online terrorist-related content within one hour. In theory, "nerd harder" rules encourage technologists to innovate even more. In practice, "nerd harder" laws either eliminate user-generated content outright, or lead to a single industrywide standard dictated by legal considerations, stifling innovation.

Conclusion

In the 1990s, Lawrence Lessig (now of Harvard Law) distinguished "East Coast code," legislation produced by regulators, from "West Coast code," software produced by technologists. Both types of code can regulate online behavior, but technologists often prefer West Coast code because it is more adaptable and usually spurs, rather than inhibits, additional innovation.

Section 230 is an example of East Coast code—but with the twist that it prevents other East Coast code from controlling user-generated content. Section 230 has literally sidelined thousands of state and local regulators from imposing their code on the Internet.

All that is at risk now. Across the globe and in the U.S., regulators are aggressively attempting to shape the Internet to their specifications. This regulatory frenzy will take more control out of the hands of the West Coast coders and put it into the hands of the East Coast coders.

The intervention of East Coast coders will not help the Internet reach its technological and social potential. Section 230 gave technologists the freedom to develop the modern Internet. Not every Internet service has embraced Cox & Wyden's hope that they would voluntarily undertake socially responsible content moderation. Still, Section 230 demonstrated that protecting the freedom to code allows technologists to develop amazing solutions that produce extraordinary social utility and, concomitantly, extraordinary social and private financial benefits. Regulators should look for more opportunities to do that—starting by protecting Section 230 from becoming the next victim of East Coast code. C

Eric Goldman (egoldman@gmail.com) is a professor at Santa Clara University School of Law, Santa Clara, CA, USA.

DOI:10.1145/3349274

Hilarie Orman

▸ Carl Landwher, Column Editor

Privacy and Security

Online Voting: We Can Do It! (We Have To)

Seeking to make online voting more secure than today's flawed paper systems.

WHEN IT COMES to elections, nine of 10 security experts agree that cellulose is safer than electricity.[a] The best way to vote, they say, is to take the horse-and-buggy down to town on election day, mark up a paper ballot, and put it in a ballot box. The very thought of online voting is anathema. Yet, the status quo in voting is hardly secure, and online voting is inevitable. The agenda for concerned security experts should be to assure online voting is more secure than paper voting. If 60 years of computer security research cannot yield the solution, then it has been going in the wrong direction.

The nay-saying experts are not Luddites, exactly. They recognize the evolution of the field of computer security has followed an arc with its greatest successes in practical solutions to common problems (virus scanning, public key protocols for website verification and privacy, constant inspection and testing for system security errors) rather than in complex, infrequent use cases. Election requirements for assuring a democratically elected government go beyond the usual needs of commerce and communication.

Election officials check voters' identification documentation during midterm election voting at Key School, Arlington, VA, USA, circa November 2018.

PHOTO BY ROB CRANDALL/SHUTTERSTOCK.COM

Even as voting technology is recognized as critical infrastructure,[b] the cost of paper elections burdens government. A small number of local officials are the arbiters of how much to spend on ballots and machines and IT infrastructure. Their solutions are often draconian, discriminatory, and unsafe. In contrast, Internet access through smartphones will be nearly universal and could have a cost-per-vote of essentially zero. To make voting as easy as possible for the greatest number of citizens, we must take the condemnation of online voting as a challenge rather than a prohibition.

Paper ballots, filled out at a polling station and counted by optical scanners, have been endorsed by a number of computer scientists and voting rights groups. Although the method ranks high on transparency and auditability, it is as fraught with inequities and security problems as are the software apps on mobile devices. The only vir-

a Expert sign-on letter to Congress: Secure American elections. Discourage voters from voting online in any form—via Web, email, or fax—even in states where it is legal. Inform voters that electronically submitted ballots can be modified, copied, rerouted, or simply deleted during transmission; see https://bit.ly/2Gx4JWa.

b Statement by Secretary Jeh Johnson on the designation of election infrastructure as a critical infrastructure subsector. *DHS Archive* (2017); see https://bit.ly/2i2xNZF.

tue of paper is that large-scale fraud is arguably more difficult because either fraudsters must show up in person, or a fraudster has to approach the voter. Counteracting that advantage is the fact that in-person voting tends to suppress the votes of working people, shut-ins, and those who live in rural areas, and it incurs no small cost in running polling stations. That is why mail-in ballots are becoming cost-saving standard. Unfortunately, it lacks checks on integrity and has no guarantees of timeliness. For a relevant example of poor integrity, note that North Carolina, U.S.A., invalidated an important election[4] because of mail-in ballot fraud. Detection of fraud is hardly a cure; the election had to be conducted all over again.

Another disadvantage of traditional voting systems is their practical lack of transparency. Recounts can be conducted by examining paper ballots, but that is slow and costly, and only a few people actually get to see the ballots.

The automation of the initial counting brings into question the integrity the IT resources of the election authority's systems for handling registration and tallying. Memory cards from optical scan machines can go missing, voter registrations can get lost in processing. The public has no insight into the configuration of the resources and how they are accounted for. For numerous examples, I have to look no further than my own voting district.[3] We can do much better.

Despite the admonitions of experts, online voting is emerging in several U.S. and international initiatives; Estonia has the most aggressive effort.[1] Online votes in Estonia have been steadily growing since 2005, and the uptake may exceed 50% this year. Their system requires a USB card reader, but there is a smartphone-based app in the works. Some U.S. states allow Internet voting for overseas military personnel. West Virginia, and Denver, CO, are experimenting with a mobile voting app that includes a blockchain component.

The looming question is: Can online voting be more secure than today's flawed paper systems? I believe we have the technology pieces to reach this goal, and it is imperative to develop secure voting systems running on common mobile devices.

If the threat environment is controllable, online elections with security safeguards are possible today. For a professional society, or the governing board of a small non-profit organization, the Helios open source web-based voting system[7] offers some strong cryptographic assurances. It assures vote privacy, transparent audit, and protection against voter coercion (vote early, vote often, only the last vote counts). Helios is scalable and easy to implement. On the negative side, it can be undermined by real-world problems with voter registration, phishing attacks and malware, dirty social media tricks, and so forth. But let's take Helios as a building block with the challenge to secure it in a voting ecosystem.

A verifiable chain of trust from the voting device through to the election results publication is an absolute necessity. That chain must be robust in the face of concerted attacks on every step of the process.[6] Fraud must be detectable and close to 100% preventable. Unfortunately, mobile devices in their current state are untrustworthy. It is far too easy to introduce corrupt software that fools the voter and election officials while also corrupting the vote. The voter cannot trust the integrity of his computing device's software or firmware or hardware, nor its ability to connect to the correct server, to trust the presented ballot, to believe the vote is private, nor that the vote reached the election officials. Even if the vote is delivered correctly, voters should worry about the integrity of the server software for recording and counting. These are the challenges that should be attracting the best of our security expertise.

The key to success is the development of minimal and fully analyzed components. First, a trusted computing base on trusted hardware, verifiable software, and a public log of voter credentials and votes. Yes, those are the very things you likely did not want to see mentioned: TPM, open source, and ... blockchain! With them, a chain of trust can be built: trust in the mobile computing device, the website server, the presentation of the ballot, marking the ballot, submission of a private vote, counting the vote, and vote audit.

Can online voting be more secure than today's vote-by-mail or vote-in-person systems?

A handful of U.S. initiatives are developing precursor technology that might eventually enable secure smartphone voting, but they are directed at a simpler issue: the security of electronic voting machines and vote tabulating machines. Today, these have questionable security because they are based on commodity hardware and proprietary software. To be trustworthy, they must be based on a secure hardware and open source secure software. Within a DARPA program[2,5] for developing secure hardware and firmware, there is one grant for secure voting machines on secure hardware.

The unique challenges of online voting need to balance anonymity, authorization, and transparency. Using a blockchain for credentials solves a vexing Public Key Infrastructure (PKI) problem: The U.S. does not have a "root of trust" for election authorities, and it probably should not institute one. Each state has the right and responsibility for registering voters and conducting elections, and the individual counties (or districts) have a great deal of latitude in how they implement the processes. This means there are at least 50 root authorities. If each one has a public key, where is it advertised? What is trustworthy?

Blockchains are useful for establishing secure identities without a central authority. The election official of a state (governor, lieutenant governor, secretary of state) can issue a public key for granting election authority, and enter that key on a blockchain. There it can accumulate endorsements from other authorities: states, federal agencies, and so forth. A consensus protocol can establish trust by a preponderance of evidence. The state's public key can be used to endorse the public keys of the county election authorities and other voting districts. Key management for the more than 3,000 counties and 10,000 voting districts in the U.S.A. is a nontrivial task. The blockchain carries an immutable log of the history, revocations, authorizations, and so forth.

The multiplicity and independence of election administration regions prevents adoption of a single software base. There are a few obvious critical items for standardization into open source components. One is a two-way secure communication protocol for voters to use in establishing their voting session. The other concerns the presentation of the ballot and the voter responses. Ballots can be complicated, and the voter must be able to read the options unambiguously. A standardized and formally verifiable markup language will assure the voter's device can interpret the ballot uniquely and clearly, and verified software on the mobile device will convey the responses into a similarly unambiguous format.

How do voters know the correct software is running on their devices, how do they know their response got to the server? One needs a chain of trust established through trustworthy processors, public keys, blockchain logging, and runtime software audits. The most practical solution would include a standard trusted processing module in all mobile devices along with the minimal verified software for the trusted computing base (TCB). A "trusted path" operation on the cellphone would lock the device interactions into the trusted processor.

The TCB must include the keyboard and device display. Each manufacturer may have separate device drivers, so the variation in interfaces creates a possibly different attack surface for each device. This is another security challenge: nearly device independent hardware/firmware driver designs with common components for critical functions.

The final challenge is economic. How much will the software cost, how much would a smartphone with the trusted hardware and software cost, how much would state and county governments have to spend, and who would pay for development?

My rough estimate is that if the government required cellphones to have the secure processor and software (as user-invocable options solely for voting), the cost would be approximately $10 per unit. This would impact the affordability of the lowest-price cellphones, but the U.S. federal government could subsidize it by transferring landline taxes to a secure voting initiative. States could easily fund the software initiative through the expected savings in reduced election costs.

The unique challenges of online voting need to balance anonymity, authorization, and transparency.

Some voters will continue to need paper ballots or in-person voting at county headquarters, and mobile polling stations are needed to assist rural voters. This support can diminish over time as secure and assistive technologies develop. New opportunities for developing voting devices for the various "abilities" will develop as offshoots of device security research.

Universal, secure online voting cannot be ready by 2020. A national goal of 5% online voting for 2024 seems reasonable, with 50% by 2028. The choice is not really between online voting and paper voting, it is between risky online voting and secure online voting. C

References
1. Aasmae, K. Online voting: Now Estonia teaches the world a lesson in electronic elections. ZDNet (2019).
2. DARPA awards for secure hardware/firmware. Federal Business Opportunities (2017).
3. Davidson, L. Ben McAdams widens lead a bit over MIA love as Governor, Lt. Gov. rip Utah county for election foulups. *Salt Lake Tribune* (2018).
4. Gardner, A. N.C. board declares a new election in contested house race after the GOP candidate admitted he was mistaken in his testimony. *Washington Post* (2019).
5. Marks, J. The cybersecurity 202: Darpa has a plan to making voting machines far more secure. *Washington Post* (2019).
6. Marquardt, A., Conte, M., and Cohen, Z. Russia sought to interfere with us election systems in 2018 midterms, U.S. official says. CNN (2019).
7. Pereira, O. Internet voting with Helios, 2016.

Hilarie Orman (hilarie@purplestreak.com) is President at Purple Streak, Woodland Hills, UT, USA.

Calendar of Events

Sept. 5–7
NANOCOM '19: ACM The 6th Annual Int'l Conference on Nanoscale Computing and Communication, Dublin, Ireland, Sponsored: ACM/SIG, Contact: Tommaso Melodia, Email: melodia@ece.neu.edu

Sept. 7–10
BCB '19: 10th ACM Int'l Conference on Bioinformatics, Computational Biology and Health Informatics, Niagara Falls, NY, Sponsored: ACM/SIG, Contact: Xinghua Shi, Email: x.shi@uncc.edu

Sept. 9–13
UbiComp '19: The 2019 ACM Int'l Joint Conference on Pervasive and Ubiquitous Computing London, U.K., Co-Sponsored: ACM/SIG, Contact: Katayoun Farrahi, Email: k.farrahi@soton.ac.uk

Sept. 16–20
RecSys '19: 13th ACM Conference on Recommender Systems, Copenhagen, Denmark, Sponsored: ACM/SIG, Contact: Toine Bogers, Email: toinebogers@gmail.com

Sept. 17–20
HT '19: 30th ACM Conference on Hypertext and Social Media, Hof, Germany, Sponsored: ACM/SIG, Contact: Claus Atzenbeck, Email: claus@atzenbeck.de

Sept. 19–20
SAP '19: ACM Symposium on Applied Perception, Barcelona, Spain, Sponsored: ACM/SIG, Contact: Solene Neyret, Email: solene.neyret@ub.edu

Sept. 21–23
AFT '19: 1st Advances in Financial Technologies Zurich, Switzerland Sponsored: ACM/SIG Contact: Ittay Eyal Email: ittay@technion.ac.il

DOI:10.1145/3349276

Peter J. Denning

The Profession of IT

An Interview with Andrew Odlyzko on Cyber Security

Is a "Cyber Pearl Harbor" any greater a risk than a natural disaster? How shall we prioritize our preparations for a cyber disaster?

THE CYBER INSECURITIES of the Internet are widely touted as precursors of a "Cyber Pearl Harbor," that could by some reckonings mark the end of civilization. What if this is not a grave risk at all? What if the systems aspects we point at most frequently as the sources of vulnerabilities are actually assets in tamping down the risk? What if we spent more time developing our ability to be resilient rather than to provide absolute security?

Andrew Odlyzko—mathematician, cryptographer, author, and information technology analyst—has been asking these questions and has provided a thorough analysis. His contrarian ideas have provoked controversy.

I talked to him about this.

PHOTO COURTESY OF ANDREW ODLYZKO/UMN.EDU

Q: Military tensions among major powers have been escalating in the past few years. Government leaders are openly worried that a military-grade preemptive cyber attack could devastate a nation. What do you think of this?

A: As we increase our reliance on digital technologies, attackers will find networks of computers increasingly attractive targets. So yes, there will surely be a "Cyber Pearl Harbor." We know not the day nor the hour.

What we have to remember is that a devastating cyber event can result not only from hostile attacks but also from natural events such as solar coronal ejections that fry electronics on Earth. We are also subject to devastations from other events such as conventional wars, terror attacks, earthquakes, tsunamis, or superstorms. Some disasters are caused by innocent human mistakes, too, simple coding or operational errors, or unanticipated interactions of complex systems. Any of these events can lay waste to a region or country. It is impossible to prevent all these disasters. So the question must be: How do we prepare with maximum resiliency to recover rapidly? And how much of that effort should be devoted to security in the cyber realm?

Q: Given the range of possible devastating cyber disasters, what security measures would you recommend?

A: It depends very much on the nature of the organization. A business should focus on protecting the business; a government agency should focus on protecting the country.

Massive cyber attacks are certainly a threat. But it seems fairly well established that they can only be launched by sophisticated, well-resourced adversaries who have ample time to prepare. That basically means nation-states and possibly terrorist and criminal organizations. Such adversaries must be dealt with by national military and intelligence agencies and by international collaborative efforts. Just as we never expected most citizens to build personal fallout shelters, we should not expect them to acquire and manage information systems that would resist attacks by a determined large agency.

Governments rely also on strategic doctrines such as the balance between offense and defense and their ability to deter aggressive acts. Those actions obviously influence the probabilities of massive attacks.

More than anything, government agencies must concentrate on general resilience. Note that resilience is desirable in general, not just against hostile attacks. Protection against bioterrorism does not differ much from protection against natural pandemics. Similarly, restoration of computer networks is similar whether they are brought down by a geomagnetic storm, an electro-magnetic pulse from a nuclear explosion in space, or a cyber attack.

Q: But what about non-government organizations? What should they do?

A: A business or educational institution should worry primarily about the mundane attacks that affect its operations. This effort is of general value because protection against the mundane also reduces exposure to massive attacks in the Internet.

Standard measures such as antivirus software, firewalls, two-factor authentication, security training, and basic security practices are what regular enterprises should concentrate on.

All organizations should make it a priority to protect their data through regular, hard-to-corrupt backups. The ability to restore data is an essential part of resilience and recovery.

Enterprises can further increase their resiliency by participating in backup communication networks, including even amateur (ham) radio. And I could go on to list more steps of similar nature, all helpful in securing cyberspace.

Q: All the measures you have cited are standard ones. They have been advocated by security experts for decades. Why has your ACM Ubiquity essay (see https://bit.ly/2G5b76S) caused controversy?

A: The utter familiarity of this advice is a key part of my argument. We have known for decades of these methods for improving cybersecurity. They are taught widely in courses and discussed in books. They are not secret. Yet most of the damaging cyber attacks we have suffered could have been prevented by implementing those measures.

So the big questions are: Why were those steps not taken, and what has been the result? My (controversial) answer is that cybersecurity has simply not been very important. The "Cyber Pearl Harbor" scenarios are seen as far removed from day-to-day operations of civilian enterprises. What they have to deal with is regular crime and regular mistakes, similar to what they have always faced in the physical realm. There have been a few headline-grabbing cyber attacks involving theft of personal identification information from firms with large databases. These are a small percentage of all cyber attacks. These events illustrate my point. The companies involved did not consider the risk of massive theft to be important enough to invest in strong security measures. They now see that they were wrong.

More than anything, government agencies must concentrate on general resilience.

We have an online ecosystem in which crime is being kept within bounds by countermeasures by enterprises and law enforcement agencies. In almost all cases, criminals aim to steal data or money without divulging their identities or destroying systems.

As the economy and society at large increase their dependence on information technologies, crime is migrating into cyberspace. As a result, more resources are being put into cybersecurity. This is happening at a measured pace without drastic reengineering of our systems.

Q: Security experts have said that much software code is a mess of "spaghetti" that cannot be verified as correct. There is a dark industry that painstakingly searches through the tangled codes and sells its findings as "zero day exploits" on the black market. Purchasers of these exploits are able to launch surprise attacks and inflict serious damage before the victims are able to defend themselves with new patches. On what basis have you concluded that "spaghetti code" is not a great risk?

A: I have not concluded that at all. "Spaghetti code" is a risk, and is indeed continually being exploited by attackers. What I point out is that "spaghetti code" also has positive features. Attackers are seldom able to make clean penetrations that leave no traces, and when they insert their own malware, they often mess up.

Stuxnet—a virus that damaged Iranian nuclear centrifuges—is a famous example. Although attribution has been difficult, security experts have placed a strong likelihood that Stuxnet was a collaboration between the U.S. and Israel, based on the style of coding, similarity to other programs, and the variable names used. And, of course, the creators of Stuxnet did slip up fairly substantially in that it escaped into the wild from the Iranian facilities.

Q: When I grew up, operating systems were much smaller and more cleanly organized. Some early operating systems were under 50-thousand lines of code. Today's major operating systems are closing in on 100-million lines, and one of the open source Linux distributions is near 500 million. None of those systems has been formally verified. They are cited as premier examples of spaghetti code. And yet today's major operating systems are amazingly reliable compared to the old. How do you explain the rise of reliability along with the rise of complexity?

A: Much of the progress is due to the superabundance of storage space and cycles. This enables us to tolerate the bloat induced by patches and repairs, most of which are to the mass of software outside the operating system kernel. The accumulation of patches does generally make systems more reliable. Further, designers now devote a lot of resources to programs that monitor other programs and they test far more exhaustively than before. Even though there are strange states you can push systems into—which is what many hostile exploits do—those states tend not to occur in the situations that matter to regular users most of the time.

Many of the prescriptions of software engineering are violated routinely. For example, we know how to eliminate the continuing vulnerability to buffer overruns—but we have not done so. Still, progress has been substantial. Disciplined coding practices and isolation techniques such as sandboxing have been major factors improving reliability.

As the economy and society at large increase their dependence on information technologies, crime is migrating to cyberspace.

As you mentioned, we are unable to formally verify the giant operating systems we most rely on. But we can formally verify small systems, such as those needed to run reliable backup systems. Those are key to recovery, and thus to resilience.

Q: You have said that some of the still-popular older technologies for security such as firewalls are less secure. Can you say more?

A: Firewalls have been getting less effective. One reason is that more and more of the traffic is encrypted, and thus increasingly difficult for firewalls to classify. Another is that the entire digital environment of the enterprise has changed. Originally, firewalls were a good way to protect trusted internal systems from hostile penetration. Today the architecture of enterprises has changed considerably. Their systems are intertwined with those of suppliers, partners, and customers, as well as with devices owned by employees. Much computation happens in the cloud, not the local network. In this environment, security professionals have less ability to see and control what is happening. There is no well-defined security perimeter for a firewall to protect.

In addition, far more of the attacks rely on human engineering—for example, phishing, whaling, ransomware, frauds, deceptions, social engineering. Firewalls cannot stop them.

On the other hand, firewalls continue to improve. They are far more sophisticated than their early incarnations of two decades ago. They are not about to disappear.

Q: You have a reputation for taking contrarian stands on issues. This seems to result from your desire to understand whether popular claims stand on solid ground—and frequently they do not. A few years ago you challenged Metcalfe's Law that the value of a network grows with the square of the number of nodes. What was your challenge and what came of that?

A: The argument (developed in a paper with Briscoe and Tilly) was that Metcalfe's Law overestimated the value of a network. We proposed that usually a more accurate measure was given by the product of the number of nodes and the logarithm of that. This proposal has held up quite well. This leads to a more realistic view of the size of network effects for new technologies, and therefore of the prospects of new ventures.

More generally, contrary opinions do help broaden people's horizons and prepare them for the inevitable surprises. In some cases, the dominant consensus is not just wrong, but leads to substantial waste of time and resources. That is the case with the apocalyptic claims about cybersecurity. There is much talk about need for drastic action and reengineering our systems from the ground up. But this talk is not matched by actions. Technologists overestimate their chances of making big impacts with their radical proposals. There is a need for improved security technologies, but when we look at decisions that are being made, we see they implicitly assume that security is important but not urgent. This is likely to continue. I expect us to continue to make good progress in staying ahead of criminals and attackers without radical changes in Internet and operating system architectures. C

Andrew Odlyzko is a mathematician and a former head of the University of Minnesota's Digital Technology Center and of the Minnesota Supercomputing Institute, USA. He was previously a researcher and research manager at Bell Labs and AT&T Labs, USA. His recent works are available at his home page http://www.dtc.umn.edu/~odlyzko.

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, USA, 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:10.1145/3347864

David Auerbach

Viewpoint
Bitwise: A Life in Code

Data science as a paradox.

IN 1960, PHYSICIST Eugene Wigner pondered "The Unreasonable Effectiveness of Mathematics in the Natural Sciences," wondering why it was that mathematics provided the "miracle" of accurately modeling the physical world. Wigner remarked, "it is not at all natural that 'laws of nature' exist, much less that man is able to discover them." Fifty years later, artificial intelligence researchers Alon Halevy, Peter Norvig, and Fernando Pereira paid homage to Wigner in their 2009 paper "The Unreasonable Effectiveness of Data," an essay describing Google's ability to achieve higher quality search results and ad relevancy not primarily through algorithmic innovation but by amassing and analyzing orders of magnitude more data than anyone had previously. The article both summarized Google's successes to that date and presaged the jumps in "deep learning" in this decade. With sufficient data and computing power, computer-constructed models obtained through machine learning raise the possibility of performing as well if not better than human-crafted models of human behavior. And since machines can craft such models far more quickly than humans, data-driven analytics and machine learning appear to promise more efficient, accurate, and rich models—at the cost of transparency and modularity, hence why such systems are frequently seen as black boxes.

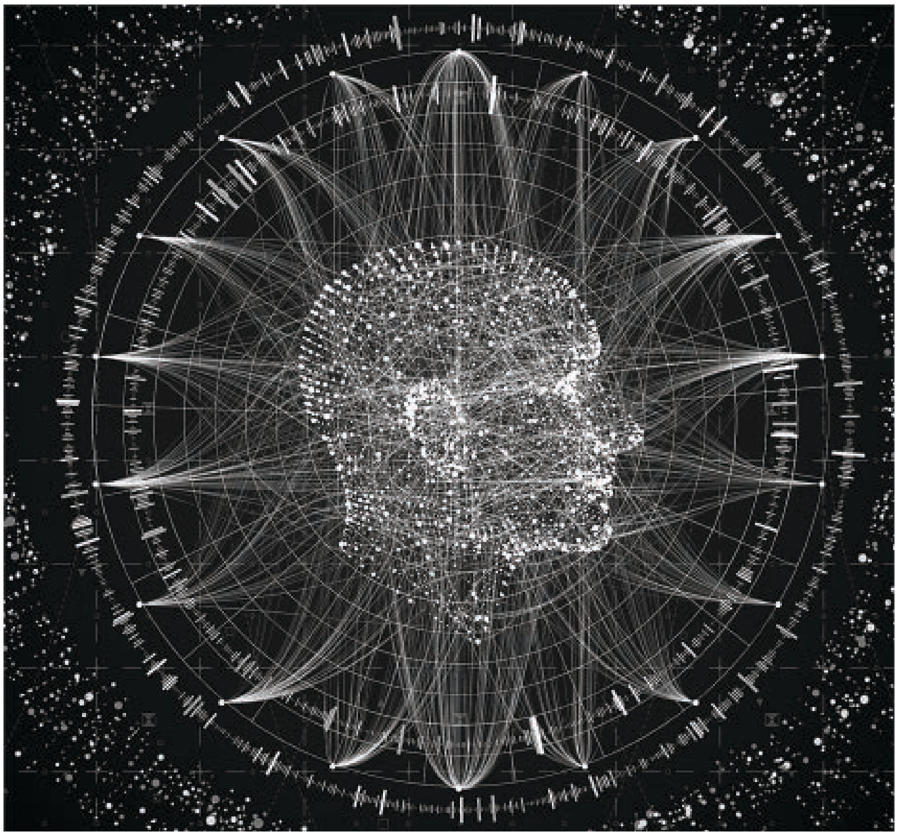


IMAGE BY GARRY KILLIAN

Ironically, Halevy, Norvig, and Pereira's insights were driven by the ineffectiveness of mathematics in the life and social sciences. It is, I hope, an undisputed contention that models of biological and human behavior have nowhere near as strong the predictive power as do physical laws. While little tenable survives of Aristotle's physics, the fourfold humoural classification of ancient Greece endures through the Jungian temperament classifying systems employed by the majority of Fortune 500 companies. Where such folk theories still prevail, there is the potential for automated computer models to do better than our own. We do not need machine learning for physical laws, only for phenomena so apparently complex we lack an unreasonably effective model of them.

I wrote my book *Bitwise: A Life in Code* (Pantheon) to chronicle my own struggle to reconcile the beautiful precision of computer science and mathematical models with the messiness of human existence. Yet the problems that I mused upon as a student of computer science and literature in the 1980s and 1990s grew far more relevant as the "datafication" of the world took place through the growth of the Internet and computing power. What is facing us, I argue, are several related phenomena:

- The perpetual inadequacy of our provisional models of human behavior, psychology, sociology, and economics.
- The tension between overly simplistic and reductionistic models versus opaque or overfitted models, and the difficulty finding a happy medium between them.
- The rush to computationally regiment these models, however inadequate they may be, so that computers can view people, groups, and other phenomena in their terms.
- The ad hoc, large-scale adoption of human- *and* computer-generated models of human behavior and psychology in the service of creating computationalized profiles and analyses of human beings.

Consequently, I suggest there is something paradoxical about "data science" as it exists today, in that it frequently begins with approximate, inaccurate models only to be faced with the choice of either reifying them or superseding them with more inscrutable models. While some methods offer a degree of scrutability and explanation, the difficulty of utilizing these methods to externally validate models arises from the very factor that enables their success: the sheer amount of data and processing being done.

Two consequences of this massive increase in data processing are a drive toward ubiquity of the models used, and an increasing human opacity to these models, whether or not such opacity is intended or inevitable. If our lives are going to be encoded (in the non-programming sense) by computers, computer science should assume reductionism, ubiquity, *and* opacity as intrinsic properties (and problems) of the models its methods generate.

Networks Reify Models

The present era of computing has been labeled with buzzwords like "big data" and "deep learning," the unifying thread among them being the centrality of enormous datasets and the creation of persistent, evolving networks to collect, store, and analyze them. I count not only machine learning networks among them, but the data stores and applications of Google, Facebook, Amazon, and myriad others. As an engineer at Google in the mid-2000s, I observed that systems, analytics, and machine learning were all fields that converged on the company's fundamental goal of making the greatest possible use of its data, in part by collecting more of it.

Inasmuch as these persistent, evolving networks rely on overt and hidden properties of their data to achieve their particular results, models are central to such networks in a way they are not to algorithms. Whether one is performing a simple MapReduce or training a neural network, the choice of which features (or signals) to analyze and how to weigh and combine them constitutes guidance toward an implicit or explicit model of the data being studied. In other words, one is always working with and toward a model that relates the network's data to its practical applications. No two such networks are alike because each is the product of the particular data on which it is built, trained, or deployed. And in the absence of some overarching coordination, we should not expect these models to be compatible with each other. Alan Perlis said, "Every program is a part of some other program and rarely fits," and the same is doubly true for models.

The explicit models employed by the largest networks today are frequently simplistic in the extreme. For Twitter, they are primarily a collection of hashtags and other keywords which serve to lump users together into overlapping categories. For Facebook, the core models include the basic personal information about users, a set of demographic microcategories, a set of six emotional reactions, and a large set of products, hobbies, and cultural objects in which people can express interest. Amazon adopted existing taxonomies of consumer products and combined them with the user information common among other large networks. National identification and reputation systems such as China's Social Credit System and India's Aadhaar extend the sort of categorization employed by Facebook to organize citizens under governmentally adjudicated labels. The results, by general agreement, are paradoxically elaborate yet simple, discrete yet haphazard. True, such models are not meant to be definitive or "scientific," but forcing users into such taxonomies results in these taxonomies carrying an increasingly prescriptive element. They thus become *ontologies* in serving to regiment our reality. It is the irony of the data age that computers, with little to no understanding of the models they are employing, are increasingly acting as primary arbiters of the ontologies employed by humans.

The specific problem computer science faces in these "data network" scenarios is that of making the data "safe" and "accurate" for the networks.

The specific problem computer science faces in these "data network" scenarios is that of making the data "safe" and "accurate" for the networks. That is, if we assume that the fit of the data to reality is imprecise and insufficient, what general purpose techniques can computer science itself offer to mitigate the inevitable flaws of the resulting models?

Answers to this question tend to display one of two opposing tendencies: on the one hand, a reductionistic transparency; on the other, a complexified opacity. In practice, networks tend to use a combination of both. Consumers and users may be asked to self-categorize themselves, being forced to choose from a shortlist of discrete options. On the other hand, statistical and machine learning methods may be used to fit humans into categories based on training data or tuned heuristics. Nevertheless, these categories are generally explicitly specified by the creators of the network performing the analysis, meaning that no matter the complexity of the model, refinement or revision of the ontology remains a manual process, as with Facebook's advertiser categories.

In other words, even machine-generated models are beholden to the strictures placed on them by humans. For example, Wisconsin's COMPAS recidivism algorithm tended to err in classifying blacks as more likely to recidivate and whites as less likely. Similarly, Amazon trained a résumé screening network that produced arbitrary and biased results, discriminating against women and making assessments based on linguistic choices rather than listed skills. While not entirely opaque, the models were evidently not fixable, as Amazon discontinued the project. I cite these two examples in the hopes of showing that these problems are sufficiently general that they should fall under the rubric of computer and data science rather than as application-specific failures.

Unsupervised learning suggests that machine learning may increasingly be able to create its own feature distinctions, which would mitigate the anthropocentric biases of human-specified categories and features, at the cost of making such distinctions more opaque to humans. If people, instead of being classified into human-specified definitions such as "young women who travel to Greece" and "middle-aged empty nesters making over $100,000 a year," are divided into machine-created categories without any such descriptors, what amount of meaning can those categories have to humans? This is the problem of opacity: if computers offer a model to which humans can be better fit, there is every likelihood that we ourselves would not be able to employ it in our lives. We would be more comprehensible to machines than to one another.

Models Become Opaque

For centuries now, there has been a stunning gap between the precision and accuracy of physical models of the world versus our folk psychological models of people. Computers have inherited and exacerbated this gap. The game *Dwarf Fortress* has an entire fluid dynamics engine built in it to manage the flows of water and lava, but its non-player character dwarves that people these environments are modeled around heuristic ideas based partly on medieval folk psychological theories. The comparative simplicity of the human models does not owe to any failing on designer Tarn Adams' part, but rather to a lack of existing definitive models of sentient behavior and the as-yet untamed complexity of the phenomena such models attempt to capture.

Inscrutibility appears to be a mark of the human.

When it comes to coding complex phenomena and human phenomena in particular, it can appear as though we face an unenviable choice between simplistic, human-crafted models based in folk ontologies, and opaque, computer-crafted models that defy human explanation.

I discuss both at length in *Bitwise*, concluding that ironically, the former, reductionistic approach is closer to what we think of as "machine" and the latter, opaque approach is closer to what we think of as "human," as we successfully operate every day with vague, ambiguously underspecified systems of which none of us can give a complete or accurate account, natural language chief among them. Such human-employed systems are interpretable yet inscrutable, in that one can understand the potential purpose of saying a certain thing in a certain situation and yet remain unable to make predictions, unable to determine whether a person is about to say "It is raining" when it is raining, even if their saying so does not come as any surprise. There is no indication that the ontologies around which these systems are based have any ultimate scientific validity; what it is to be "raining" is vague and underspecified. If we are treating each other as black boxes, it is a tall order to ask computers to be transparent in their attempts to model and predict human behavior. It is a brave ideal, but one that demands honesty around its infeasibility. Inscrutability appears to be a mark of the human.

Our failure to translate our own inscrutable world models into computational terms is the primary driver of the need for the two kinds of inadequate models specified here. Of course, it is not as though these human systems are complete or accurate: few would say natural language is the ideal mechanism for expressing truths about the world. It just happens to be our shared, mutually comprehensible mechanism. If we are to cope with inscrutability in our machine-generated models, they must improve on these complex, human-used models. They must become more accurate and complete in describing phenomena while remaining comprehensible—though not necessarily wholly scrutable—to humans.

Conclusion

"What is a man so made that he can understand number, and what is number so made that a man can understand it?" This, according to Seymour Papert, was the question that guided Warren McCulloch's life as he made some of the earliest steps, alongside Alan Turing and Norbert Wiener, toward a theory of artificial intelligence in the middle of the 20th century. Today we could invert the question: "What is data so made that it can represent the human, and what are humans so made that they can present themselves in data?" If the *quantity* of data and data processing is a key differentiator between provisional success and failure in the models created and utilized by computers, then the problem of inscrutability seems unavoidable to me. Rather, the more tenable problem may be one of *synchronizing* interpretability between machine networks and humans, so that even though we each are black boxes, humans can still tune and correct machines—and vice versa. C

David Auerbach (david@davidauerba.ch) is the author of *Bitwise: A Life in Code* (Pantheon). He previously worked as a software engineer at Google and Microsoft after graduating from Yale University.

DOI:10.1145/3347446

Article development led by acmqueue
queue.acm.org

Software reuse is finally here but comes with risks.

BY RUSS COX

Surviving Software Dependencies

FOR DECADES, DISCUSSION of software reuse was more common than actual software reuse. Today, the situation is reversed: developers reuse software written by others every day, in the form of software dependencies, and the situation goes mostly unexamined.

My background includes a decade of working with Google's internal source code system, which treats software dependencies as a first-class concept,[17] as well as developing support for dependencies in the Go programming language.[2]

Software dependencies carry with them serious risks that are too often overlooked. The shift to easy, fine-grained software reuse has happened so quickly that we do not yet understand the best practices for choosing and using dependencies effectively, or even for deciding when they are appropriate and when not. The purpose of this article is to raise awareness of the risks and encourage more investigation of solutions.

In software development today, a *dependency* is additional code a programmer wants to call. Adding a dependency avoids repeating work: designing, testing, debugging, and maintaining a specific unit of code. In this article, that unit of code is referred to as a *package*; some systems use the terms *library* and *module* instead.

Taking on externally written dependencies is not new. Most programmers have at one point in their careers had to go through the steps of manually installing a required library, such as C's PCRE or zlib; C++'s Boost or Qt; or Java's JodaTime or JUnit. These packages contain high-quality, debugged code that required significant expertise to develop. For a program that needs the functionality provided by one of these packages, the tedious work of manually downloading, installing, and updating the package is easier than the work of redeveloping that functionality from scratch. The high fixed costs of reuse, however, mean manually reused packages tend to be big; a tiny package would be easier to reimplement.

A *dependency manager* (a.k.a. *package manager*) automates the downloading and installation of dependency packages. As dependency managers make individual packages easier to download and install, the lower fixed costs make smaller packages economical to publish and reuse. For example, the Node.js dependency manager NPM provides access to more than 750,000 packages. One of them, escape-string-regexp, consists of a single function that escapes regular expression operators in its input. The entire implementation is:

```
var matchOperatorsRe =
   /[|\\{}()[\]^$+*?.]/g;
module.exports = function (str) {
 if (typeof str !== 'string') {
   throw new TypeError(
      'Expected a string');
 }
 return str.replace(
    matchOperatorsRe, '\\$&');
};
```

Before dependency managers, publishing an eight-line code library would have been unthinkable: too much overhead for too little benefit. NPM, however, has driven the overhead approximately to zero, with the result that nearly trivial functionality can be packaged and reused. In late April 2019, the escape-string-regexp package was explicitly depended upon by almost a thousand other NPM packages, not to mention all the packages developers write for their own use and don't share.

Dependency managers now exist for essentially every programming language: Maven Central (Java), NuGet (.NET), Packagist (PHP), PyPI (Python), and RubyGems (Ruby) each host more than 100,000 packages. The arrival of this kind of fine-grained, widespread software reuse is one of the most consequential shifts in software development over the past two decades. And if we are not more careful, it will lead to serious problems.

IMAGE BY T. EMIL

What Could Go Wrong?

A package, for this discussion, is code downloaded from the Internet. Adding a package as a dependency outsources the work of developing that code—designing, writing, testing, debugging, and maintaining—to someone else on the Internet, often unknown to the programmer. Using that code exposes the program to all the failures and flaws in the dependency. The program's execution now literally *depends* on code downloaded from this stranger on the Internet. Presented this way, it sounds incredibly unsafe. Why would anyone do this?

Because it's easy, it seems to work, everyone else is doing it, and, most importantly, it seems like a natural continuation of age-old established practice. But there are important differences that are being ignored.

Decades ago, most developers trusted others to write the software they depended on, such as operating systems and compilers. That software was purchased from known sources, often with some kind of support agreement. There was still a potential for bugs or outright mischief,[20] but at least the developers knew who they were dealing with and usually had commercial or legal recourses available.

The phenomenon of open source software, distributed at no cost over the Internet, has displaced many of those earlier software purchases. When reuse was difficult, there were fewer projects publishing reusable code packages. Even though their licenses typically disclaimed, among other things, any "implied warranties of merchantability and fitness for a particular purpose," the projects built up well-known reputations that often factored heavily into people's decisions about which to use. The commercial and legal support for trusting software sources was replaced by reputational support.

Many common early packages still enjoy good reputations: consider BLAS (published in 1979), Netlib (1987), libjpeg (1991), LAPACK (1992), HP STL (1994), and zlib (1995).

Dependency managers have scaled down this open source code reuse model. Now, developers can share code at the granularity of individual functions consisting of tens of lines of code. This is a major technical accomplishment. Myriad packages are available, and writing code can involve a large number of them, but the commercial, legal, and reputational support mechanisms for trusting the code have not carried over. Developers trust more code with less justification for doing so.

The cost of adopting a bad dependency can be viewed as the sum, over all possible bad outcomes, of the cost of each bad outcome multiplied by its probability of happening (risk), as shown in the equation.

$$\text{expected cost} = \sum_{b \in \text{bad outcomes}} \text{cost}(b) \times \text{probability}(b)$$

The context in which a dependency will be used determines the cost of a bad outcome. At one end of the spectrum is a personal hobby project, where the cost of most bad outcomes is near zero: you are just having fun, bugs have no real impact other than wasting time, and even debugging can be fun. So, the risk probability almost doesn't matter—it's being multiplied by a failure cost of almost zero. At the other end of the spectrum is production software that must be maintained for years. Here, the cost of a bug in a dependency can be very high: servers may go down, sensitive data may be divulged, customers may be harmed, or companies may fail. High failure costs make it much more important to estimate and then reduce any risk of a serious failure.

Developers trust more code with less justification for doing so.

No matter what the expected cost, experiences with larger dependencies suggest some approaches for estimating and reducing the risks of adding a software dependency. Better tooling is likely needed to help reduce the costs of these approaches, much as dependency managers have focused to date on reducing the costs of downloading and installation.

Inspect the Dependency

You would not hire a software developer you have never heard of and know nothing about. You would learn more about the person first: check references, conduct a job interview, run background checks, and so on. Before you depend on a package found on the Internet, it is similarly prudent to learn a bit about it first.

A basic inspection can provide a sense of how likely you are to run into problems trying to use this code. If the inspection reveals likely minor problems, you can take steps to prepare for or perhaps avoid them. If the inspection reveals major problems, it may be best not to use the package; maybe you will find a more suitable one, or maybe you need to develop one yourself. Remember that open source packages are published by their authors in the hope they will be useful but with no guarantee of usability or support. In the middle of a production outage, you will be the one debugging the package. As the original GNU General Public License warned, "The entire risk as to the quality and performance of the program is with you. Should the program prove defective, you assume the cost of all necessary servicing, repair or correction."[7]

The following are some considerations when inspecting a package and deciding whether to depend on it:

Design. Is the documentation clear? Does the API have a clear design? If the authors can explain the package's API and its design well in the documentation, that increases the likelihood they have explained the implementation well to the computer in the source code. Writing code using a clear, well-designed API is also easier, faster, and hopefully less error-prone. Have the authors documented what they expect from client code in order to make future upgrades compatible? (Examples include the C++[23] and Go[8] compatibility documents.)

Code quality. Is the code well written? Read some of it. Does it look like the authors have been careful, conscientious, and consistent? Does it look like code you would want to debug? You may need to.

Develop your own systematic ways to check code quality. For example, something as simple as compiling a C or C++ program with important compiler warnings enabled (for example, –Wall) can give you a sense of how seriously the developers work to avoid various undefined behaviors. Recent languages such as Go, Rust, and Swift use an *unsafe* keyword to mark code that violates the type system; look to see how much unsafe code there is. More advanced semantic tools such as Infer[6] or SpotBugs[19] are helpful, too. Linters are less helpful: you should ignore rote suggestions about topics such as brace style and focus instead on semantic problems.

Keep an open mind about unfamiliar development practices. For example, the SQLite library ships as a single 200,000-line C source file and a single 11,000-line header called the *amalgamation*. The sheer size of these files should raise an initial red flag, but closer investigation would turn up the actual development source code, a traditional file tree with more than 100 C source files, tests, and support scripts. It turns out the single-file distribution is built automatically from the original sources and is easier for end users, especially those without dependency managers. (The compiled code also runs faster, because the compiler can see more optimization opportunities.)

Testing. Does the code have tests? Can you run them? Do they pass? Tests establish the code's basic functionality is correct, and they signal the developer is serious about keeping it correct. For example, the SQLite development tree has an incredibly thorough test suite with more than 30,000 individual test cases, as well as developer documentation explaining the testing strategy.[10] On the other hand, if there are few tests or no tests, or if the tests fail, that's a serious red flag. Future changes to the package are likely to introduce regressions that could easily have been caught. If you insist on tests in code you write (you do, right?), you should insist on tests in code you outsource to others.

Assuming the tests exist, run, and pass, you can gather more information by running them with runtime instrumentation such as code coverage analysis, race detection,[16] memory-allocation checking, and memory-leak detection.

Debugging. Find the package's issue tracker. Are there many open bug reports? How long have they been open? Are there many fixed bugs? Have any bugs been fixed recently? If you see lots of open issues about what look like real bugs, especially if they have been open for a long time, that's not a good sign. On the other hand, if the closed issues show that bugs are rarely found and promptly fixed, that's great.

Maintenance. Look at the package's commit history. How long has the code been actively maintained? Is it actively maintained now? Packages that have been actively maintained for an extended amount of time are more likely to continue to be maintained. How many people work on the package? Many packages are personal projects that developers create and share for fun in their spare time. Others are the result of thousands of hours of work by a group of paid developers. In general, the latter kind of package is more likely to have prompt bug fixes, steady improvements, and general upkeep.

On the other hand, some code really is "done." For example, NPM's escape-string-regexp, shown earlier, may never need to be modified again.

Usage. Do many other packages depend on this code? Dependency managers can often provide statistics about usage, or you can use a Web search to estimate how often others write about using the package. More users should at least mean more people for whom the code works well enough, along with faster detection of new bugs. Widespread usage is also a hedge against the question of continued maintenance; if a widely used package loses its maintainer, an interested user is likely to step forward.

For example, libraries such as PCRE or Boost or JUnit are incredibly widely used. That makes it more likely—although certainly not guaranteed—that bugs you might otherwise run into have already been fixed, because others ran into them first.

Security. Will you be processing untrusted inputs with the package? If so, does it seem to be robust against malicious inputs? Does it have a history of security problems listed in the NVD (National Vulnerability Database)?[13]

For example, in 2006 when Jeff Dean and I started work on Google Code Search[5]—grep over public source code—the popular PCRE regular expression library seemed like an obvious choice. In an early discussion with Google's security team, however, we learned that PCRE had a history of problems such as buffer overflows, especially in its parser. We could have learned the same by searching for PCRE in the NVD. That discovery did not immediately cause us to abandon PCRE, but it did make us think more carefully about testing and isolation.

Licensing. Is the code properly licensed? Does it have a license at all? Is the license acceptable for your project or company? A surprising fraction of projects on GitHub have no clear license. Your project or company may impose further restrictions on the allowed licenses of dependencies. For example, Google disallows the use of code licensed under AGPL-like licenses (too onerous), as well as WTFPL-like licenses (too vague).[9]

Dependencies. Does the code have

dependencies of its own? Flaws in indirect dependencies are just as bad for your program as flaws in direct dependencies. Dependency managers can list all the transitive dependencies of a given package, and each of them should ideally be inspected as described here. A package with many dependencies incurs additional inspection work, because those same dependencies incur additional risk that needs to be evaluated.

Many developers have never looked at the full list of transitive dependencies of their code and do not know what they depend on. For example, the NPM user community discovered in March 2016 that many popular projects—including Babel, Ember, and React—all depended indirectly on a tiny package called left-pad, consisting of a single eight-line function body. They discovered this when the author of left-pad deleted that package from NPM, inadvertently breaking most Node.js users' builds.[22] And left-pad is hardly exceptional in this regard. For example, 30% of the 750,000 packages published on NPM depend—at least indirectly—on escape-string-regexp. Adapting Leslie Lamport's observation about distributed systems, a dependency manager can easily create a situation in which the failure of a package you did not even know existed can render your own code unusable.

Test the Dependency

The inspection process should include running a package's own tests. If the package passes the inspection and you decide to make your project depend on it, the next step should be to write new tests focused on the functionality needed by your application. These tests often start out as short stand-alone programs written to ensure you can understand the package's API and that it does what you think it does. (If you can't or it doesn't, turn back now!) It is worth making the extra effort to turn those programs into automated tests that can be run against newer versions of the package. If you find a bug and have a potential fix, you will want to be able to rerun these project-specific tests easily, to ensure the fix did not break anything else.

It is especially worth exercising the likely problem areas identified by the basic inspection. For Code Search, we knew from past experience that PCRE sometimes took a long time to execute certain regular expression searches. The initial plan was to have separate thread pools for "simple" and "complicated" regular expression searches. One of the first tests was a benchmark comparing pcregrep with a few other grep implementations. For one basic test case, pcregrep was 70 times slower than the fastest grep available, so we started to rethink the plan to use PCRE. Even though PCRE was eventually dropped entirely, that benchmark remains in the code base today.

Abstract the Dependency

Depending on a package is a decision likely to be revisited later. Perhaps updates will take the package in a new direction. Perhaps serious security problems will be found. Perhaps a better option will come along. For all these reasons, it is worth the effort to make it easy to migrate your project to a new dependency.

If the package will be used from many places in your project's source code, migrating to a new dependency would require making changes to all those different source locations. Worse, if the package will be exposed in your own project's API, migrating to a new dependency would require making changes in all the code calling your API, which you might not control. To avoid these costs, it makes sense to define an interface of your own, along with a thin wrapper implementing that interface using the dependency. Note that the wrapper should include only what your project needs from the dependency, not everything the dependency offers. Ideally, that allows you to substitute a different, equally appropriate dependency later, by changing only the wrapper. Migrating your per-project tests to use the new interface will test the interface and wrapper implementation, as well as making it easy to test any potential replacements for the dependency.

For Code Search, we developed an abstract Regexp class that defined the interface Code Search needed from any regular expression engine. Then we wrote a thin wrapper around PCRE implementing that interface. The indirection made it easy to test alternate libraries, and it prevented accidentally introducing knowledge of PCRE internals into the rest of the source tree. That in turn ensured it would be easy to switch to a different dependency if needed.

Isolate the Dependency

Isolating a dependency at runtime may also be appropriate in order to limit the possible damage caused by bugs. For example, Google Chrome allows users to add dependencies—extension code—to the browser. When Chrome launched in 2008, it introduced the critical feature (now standard in all browsers) of isolating each extension in a sandbox running in a separate operating-system process.[18]

An exploitable bug in a badly written extension therefore did not automatically have access to the entire memory of the browser itself and could be stopped from making inappropriate system calls.[12] For Code Search, until we dropped PCRE entirely, the plan was to isolate at least the PCRE parser in a similar sandbox. Today, another option would be a lightweight hypervisor-based sandbox such as gVisor.[11] Isolating dependencies reduces the associated risks of running that code.

Even with these examples and oth-

er off-the-shelf options, runtime isolation of suspect code is still too difficult and rarely done. True isolation would require a completely memory-safe language, with no escape hatch into untyped code. That's challenging not just in entirely unsafe languages such as C and C++, but also in languages that provide restricted unsafe operations, such as Java when including JNI (Java Native Interface), or Go, Rust, and Swift when including their "unsafe" features. Even in a memory-safe language such as JavaScript, code often has access to far more than it needs. In November 2018, the latest version of the NPM package event-stream, which provided a functional streaming API for JavaScript events, was discovered to contain obfuscated malicious code that had been added 2.5 months earlier. The code, which harvested large Bitcoin wallets from users of the Copay mobile app, was accessing system resources entirely unrelated to processing event streams.[1] One of many possible defenses to this kind of problem would be to better restrict what dependencies can access.

Even after all that work, you are not done tending your dependencies. It's important to continue to monitor them and perhaps even re-evaluate your decision to use them.

Avoid the Dependency

If a dependency seems too risky and you can't find a way to isolate it, the best answer may be to avoid it entirely, or at least to avoid the parts you have identified as most problematic.

For example, as we better understood the risks and costs associated with PCRE, our plan for Google Code Search evolved from "use PCRE directly," to "use PCRE but sandbox the parser," to "write a new regular expression parser but keep the PCRE execution engine," to "write a new parser and connect it to a different, more efficient open source execution engine." Later we rewrote the execution engine as well, so that no dependencies were left, and we open sourced the result: RE2.[4]

If you need only a tiny fraction of a dependency, the simplest solution may be to make a copy of what you need (preserving appropriate copyright and other legal notices, of course). You are taking on responsibility for fixing bugs, maintenance, and so on, but you are also completely isolated from the larger risks. The Go developer community has a proverb about this: "A little copying is better than a little dependency."[14]

Upgrade the Dependency

For a long time, the conventional wisdom about software was, "If it ain't broke, don't fix it." Upgrading carries a chance of introducing new bugs; without a corresponding reward—such as a new feature you need—why take the risk? This analysis ignores two costs. The first is the cost of the eventual upgrade. In software, the difficulty of making code changes does not scale linearly: making 10 small changes is less work and easier to get right than making one equivalent large change. The second is the cost of discovering already-fixed bugs the hard way. Especially in a security context, where known bugs are actively exploited, every day you wait is another day that attackers can break in.

For example, consider what happened at Equifax in 2017, as recounted by executives in detailed Congressional testimony.[21] On March 7, a new vulnerability in Apache Struts was disclosed, and a patched version was released. On March 8, Equifax received a notice from US-CERT (United States Computer Emergency Readiness Team) about the need to update any uses of Apache Struts. Equifax ran source code and network scans on March 9 and March 15, respectively; neither scan turned up a particular group of public-facing Web servers. On May 13, attackers found the servers that Equifax's security teams could not. They used the Apache Struts vulnerability to breach Equifax's network and then steal detailed personal and financial information about 148 million people over the next two months. Equifax finally noticed the breach on July 29 and publicly disclosed it on September 4. By the end of September, Equifax's CEO, CIO, and

CSO had all resigned, and a Congressional investigation was underway.

Equifax's experience drives home the point that although dependency managers know the versions they are using at build time, other arrangements must be made to track that information through the production deployment process. For the Go language, we are experimenting with automatically including a version manifest in every binary, so that deployment processes can scan binaries for dependencies that need upgrading. Go also makes that information available at runtime, so that servers can consult databases of known bugs and self-report to monitoring software when they are in need of upgrades.

Upgrading promptly is important, but it means adding new code to your project, which should mean updating your evaluation of the risks of using the dependency based on the new version. At minimum, you would want to skim the diffs showing the changes being made from the current version to the upgraded versions, or at least read the release notes, to identify the most likely areas of concern in the upgraded code. If a lot of code is changing, so that the diffs are difficult to digest, you can incorporate that fact into your risk-assessment update.

You will also want to rerun the tests you have written that are specific to your project, to ensure the upgraded package is at least as suitable for the project as the earlier version. Rerunning the package's own tests also makes sense. If the package has its own dependencies, it is entirely possible that your project's configuration uses versions of those dependencies (either older or newer ones) different from those used by the package's authors. Running the package's own tests can quickly identify problems specific to your configuration.

Again, upgrades should not be completely automatic. You must verify the upgraded versions are appropriate for your environment before deploying them.[3]

If your upgrade process includes rerunning the integration and qualification tests you have already written for the dependency, so that you are likely to identify new problems before they reach production, then in most cases delaying an upgrade is riskier than upgrading quickly.

If a dependency seems too risky and you can't find a way to isolate it, the best answer may be to avoid it entirely, or at least to avoid the parts you have identified as most problematic.

The window for security-critical upgrades is especially short. In the aftermath of the Equifax breach, forensic security teams found evidence that attackers (perhaps different ones) had successfully exploited the Apache Struts vulnerability on the affected servers on March 10, only three days after it was publicly disclosed, but they had run only a single `whoami` command.

Watch Your Dependencies

Even after all that work, you are not done tending your dependencies. It's important to continue to monitor them and perhaps even re-evaluate your decision to use them.

First, ensure you keep using the specific package versions you think you are. Most dependency managers now make it easy or even automatic to record the cryptographic hash of the expected source code for a given package version and then to check that hash when redownloading the package on another computer or in a test environment. This ensures your build uses the same dependency source code you inspected and tested. These kinds of checks prevented the `event-stream` attacker, described earlier, from silently inserting malicious code in the already-released version 3.3.5. Instead, the attacker had to create a new version, 3.3.6, and wait for people to upgrade (without looking closely at the changes).

It is also important to watch for new indirect dependencies creeping in. Upgrades can easily introduce new packages upon which the success of your project now depends. They deserve your attention as well. In the case of `event-stream`, the malicious code was hidden in a different package, `flatmap-stream`, which the new `event-stream` release added as a new dependency.

Creeping dependencies can also affect the size of your project. During the development of Google's Sawzall[15]—a JIT'ed logs processing language—the authors discovered at various times that the main interpreter binary contained not just Sawzall's JIT but also (unused) PostScript, Python, and JavaScript interpreters. Each time, the culprit turned out to be unused dependencies declared by some library Sawzall did depend on, combined with the fact that Google's build system eliminated any manual effort needed to start using a new dependency. This kind of error is the reason the Go language makes importing an unused package a compile-time error.

Upgrading is a natural time to revisit the decision to use a dependency that's changing. It's also important to periodically revisit any dependency that *isn't* changing. Does it seem plausible that there are no security problems or other bugs to fix? Has the project been abandoned? Maybe it's time to start planning to replace that dependency.

It's also important to recheck the security history of each dependency. For example, Apache Struts disclosed different major remote code execution vulnerabilities in 2016, 2017, and 2018. Even if you have a list of all the servers that run it and update them promptly, that track record might make you rethink using it at all.

Conclusion

Software reuse is finally here, and its benefits should not be understated. It has brought an enormously positive transformation for software developers. Even so, we have accepted this transformation without completely thinking through the potential consequences. The old reasons for trusting dependencies are becoming less valid at exactly the same time there are more dependencies than ever.

The kind of critical examination of specific dependencies outlined in this article is a significant amount of work and remains the exception rather than the rule. It's unlikely that any developers actually make the effort to do this for every possible new dependency. I have done only a subset of them for a subset of my own dependencies. Most of the time the entirety of the decision is, "Let's see what happens." Too often, anything more than that seems like too much effort.

The Copay and Equifax attacks are clear warnings of real problems in the way software dependencies are consumed today. We should not ignore the warnings. Here are three broad recommendations:

1. *Recognize the problem.* If nothing else, this article hopefully convinced you that there is a problem here worth addressing. We need many people to focus significant effort on solving it.

2. *Establish best practices for today.* Best practices are needed for managing dependencies using what is available today. This means working out processes that evaluate, reduce, and track risk, from the original adoption decision through production use. In fact, just as some engineers specialize in testing, others may need to specialize in managing dependencies.

3. *Develop better dependency technology for tomorrow.* Dependency managers have essentially eliminated the cost of downloading and installing a dependency. Future development efforts should focus on reducing the cost of the kind of evaluation and maintenance necessary to use a dependency. For example, package-discovery sites might work to find more ways to allow developers to share their findings. Build tools should, at the least, make it easy to run a package's own tests. More aggressively, build tools and package-management systems could also work together to allow package authors to test new changes against all public clients of their APIs. Languages should also provide easy ways to isolate a suspect package.

There is a lot of good software out there. Let's work together to find out how to reuse it safely.

Related articles on queue.acm.org

The Calculus of Service Availability
Ben Treynor, Mike Dahlin, Vivek Rau, and Betsy Beyer
https://queue.acm.org/detail.cfm?id=3096459

Tracking and Controlling Microservice Dependencies
Silvia Esparrachiari, Tanya Reilly, and Ashleigh Rentz
https://queue.acm.org/detail.cfm?id=3277541

Thou Shalt Not Depend on Me
Tobias Lauinger, Abdelberi Chaabane, and Christo B. Wilson
https://queue.acm.org/detail.cfm?id=3205288

References

1. Baldwin, A. Details about the event-stream incident. The npm Blog (Nov. 2018); https://bit.ly/2DRjySJ
2. Cox, R. Go & Versioning, 2018; https://research.swtch.com/vgo.
3. Cox, R. The principles of versioning in Go. GopherCon Singapore (May 2018); https://www.youtube.com/watch?v=F8nrpe0XWRg.
4. Cox, R. RE2: A principled approach to regular expression matching. Google Open Source Blog (Mar. 2010); https://bit.ly/2XoLFzC.
5. Cox, R. Regular expression matching with a trigram index or how Google Code Search worked. Swtch.com (Jan. 2012); https://swtch.com/~rsc/regexp/regexp4.html.
6. Facebook. Infer: A tool to detect bugs in Java and C/C++/Objective-C code before it ships; https://fbinfer.com/.
7. GNU Project. GNU General Public License, version 1, 1989; https://www.gnu.org/licenses/old-licenses/gpl-1.0.html.
8. Go Project. Go 1 and the future of Go programs, 2013; https://golang.org/doc/go1compat.
9. Google Open Source. Using third-party licenses; https://opensource.google.com/docs/thirdparty/licenses/#banned.
10. Hipp, D. R. How SQLite is tested; https://www.sqlite.org/testing.html.
11. Lacasse, N., Open-sourcing gVisor, a sandboxed container runtime. Google Cloud (May 2018); http://bit.ly/2wzA84D.
12. Langley, A. Chromium's seccomp sandbox. ImperialViolet (Aug. 2009); https://www.imperialviolet.org/2009/08/26/seccomp.html.
13. National Institute of Standards and Technology. National Vulnerability Database—Search and Statistics; https://nvd.nist.gov/vuln/search.
14. Pike, R. Go Proverbs, 2015; https://go-proverbs.github.io/.
15. Pike, R., Dorward, S., Griesemer, R. and Quinlan, S. Interpreting the data: Parallel analysis with Sawzall. *Scientific Programming J. 13*, 4 (2005), 277–298; https://doi.org/10.1155/2005/962135.
16. Potapenko, A. Testing Chromium: ThreadSanitizer v2, a next-gen data race detector. Chromium Blog (Apr. 2014); http://bit.ly/2WN29oO.
17. Potvin, R., Levenberg, J. Why Google stores billions of lines of code in a single repository. *Commun. ACM 59*, 7 (July 2016), 78–87; https://doi.org/10.1145/2854146.
18. Reis, C. Multi-process architecture. Chromium Blog (Sept. 2008); https://blog.chromium.org/2008/09/multi-process-architecture.html.
19. SpotBugs: Find bugs in Java programs; https://spotbugs.github.io/.
20. Thompson, K. Reflections on trusting trust. *Commun. ACM 27*, 8 (Aug. 1984), 761–763; https://doi.org/10.1145/358198.358210.
21. U.S. House of Representatives Committee on Oversight and Government Reform. The Equifax Data Breach, Majority Staff Report, 115th Congress (Dec. 2018); http://bit.ly/2Gf53IJ.
22. Willis, N. A single Node of failure. LWN.net (Mar. 2016); https://lwn.net/Articles/681410/.
23. Winters, T. SD-8: Standard library compatibility, C++ standing document, 2018; http://bit.ly/2QNhT5k.

Russ Cox leads the development of the Go programming language at Google, with a current focus on improving the security and reliability of using software dependencies. With Jeff Dean, he created Google Code Search. He worked for many years on the Plan 9 from Bell Labs operating system.

DOI:10.1145/3345626

Article development led by acmqueue
queue.acm.org

From tectonic plate to F-16.

BY TOM KILLALEA

Velocity in Software Engineering

SOFTWARE ENGINEERING IS necessary in all modern companies, but software engineers are expensive and in very limited supply. So naturally there's a lot of interest in the increase of velocity from existing software-engineering investments. In most cases, software engineering is a team activity, with breakthroughs typically achieved through many small steps by a web of collaborators. Good ideas tend to be abundant, though execution at high velocity is elusive. The good news is that velocity is controllable; companies can invest systematically to increase it.

Velocity compounds. It's also habit-forming; high-velocity teams become habituated to a higher bar. When velocity stalls, high contributors creatively seek ways to reestablish high velocity, but if external forces prolong the stall, soon they will want to join another team that has the potential for high velocity. High velocity is addictive and bar-raising.

Direction. Velocity is a function of direction and speed; you can't focus on only one of these. Of the two, direction is more easily overlooked. The most common reason that projects fail, however, is that the team was building the wrong thing. As Thomas Merton more eloquently put it, "People may spend their whole lives climbing the ladder of success only to find, once they reach the top, that the ladder is leaning against the wrong wall."

Amazon's Working Backwards product-development process seeks to compensate for the difficulty of determining direction (that is, predicting product/market fit). The explicit artifacts of the Working Backwards process—a press release and an FAQ—have been widely discussed,[8] and inherent in the process is the clear identification of who the customers are, then working backward from their needs to a product definition that would viably meet those needs. Frequently it's about paying attention to the voice of the customer, or, as Intuit cofounder Scott Cook put it, "Success is not delivering a feature; success is learning how to solve a customer's problem." Teams often lament their customers use only 20% of what they shipped. Ideally, we would like to listen to customers and meet their needs while shipping only the 20% that most interests them.

Even for the best listeners and most visionary innovators, it's difficult to predict what customers need. Because there is some guesswork involved in choosing a direction, flexibility and course correcting become crucial. Flexibility might show up as openness, maximizing the rate of experimentation, learning quickly, reducing commitment to any given plan, rapidly evolving products, and distinguishing between one-way (irreversible) and two-way (reversible) doors in decision-making. As to course correcting, Amazon CEO Jeff Bezos said, "If you are good at course correcting, being wrong may be less costly than you think, whereas being slow is going to be expensive for sure."

Speed. In 2003, at a time in Amazon's history when we were particularly frustrated by our speed of software en-

IMAGE BY LEUNG CHO PAN

gineering, we turned to Matt Round, an engineering leader who was a most interesting squeaky wheel in that his team appeared to get more done than any other, yet he remained deeply impatient and complained loudly and with great clarity about how hard it was to get anything done. He wrote a six-pager that had a great hook in the first paragraph: "To many of us Amazon feels more like a tectonic plate than an F-16." That nobody responded defensively to this statement reflects well on the culture at Amazon at that time. Rather, the response was one of recognition: "He nailed us. That's us! A tectonic plate!"

Matt's paper had many recommendations that by now have become broadly adopted industrywide, including the maximization of autonomy for teams and for the services operated by those teams by the adoption of REST-style interfaces between highly decoupled components, platform standardization, removal of roadblocks or gatekeepers (high-friction bureaucracy), and continuous deployment of isolated components. He also called for the extension of the definition of "complete" to include the achievement of low ongoing maintenance costs, and for an enduring performance indicator based on the percentage of their time that software engineers spent building rather than doing other tasks. Builders want to build, and Matt's timely recommendations influenced the forging of Amazon's technology brand as "the best place where builders can build."

There have been many attempts to directly observe the velocity of software teams, but measuring such velocity is notoriously difficult. To compensate, companies can use engagement surveys to ask questions relating to velocity. Such surveys have become widespread, but too often they are limited to high-level measures of employee engagement and alignment with the company's goals. Some companies use their surveys as opportunities to determine whether they are great places for builders to build at high velocity, asking software engineers questions about how much time they spend actually designing and writing software, the adequacy of their tools, the effectiveness of their processes, and the impact of technical debt.

Software engineers can be cynical. Prior to embarking on surveys with questions such as these, companies should commit to acting on the results, so those actions positively impact both current velocity and future responses to such surveys.

Autonomy. The challenge of scaling software-engineering projects so the addition of engineers results in greater throughput has been much discussed since the publication of *The Mythical Man-Month*[3] by Fred Brooks in 1975. Brooks examined the lack of increased throughput in software-engineering projects as more engineers are added and contrasted it with activities such as reaping wheat or picking cotton. He finds blame in the cost of coordination and communication.

Scalability can be improved by organizing into autonomous teams that have a high degree of internal cohesion around a specific and well-bounded context or area of responsibility. Teams, and the services that they're accountable for, expose application programming interfaces (APIs) to each other; in an ideal world no cross-team communication occurs since the APIs are all that are needed to interact with the business logic that is the responsibility of a team behind a remote service.[5]

The implementation details of the service are not typically shared, and there is no backdoor access to the data store on which a remote service depends. Coordination becomes unnecessary; even if a service needs to change in a backward-incompatible way, the new and old versions of the APIs will typically be made available for an overlapping period of time, so clients can migrate before the old version is deprecated. Round went so far as to argue for the measurement of crosstalk between teams in order to get an objective read on their level of independence.

Service owners can evolve and release changes at their teams' own pace, independent of and decoupled in time from other changes that may be taking place around them. Permissionless innovation, "the ability of others to create new things on top of the communications constructs that we create," as defined by IETF chair Jari Arkko,[1] can flourish. The work of identifying the seams between areas of responsibility, however, is challenging, and inevitably those seams will evolve over time. Perfect autonomy will always be illusory.

Even in a world of rapidly evolving requirements, it's OK for a team's well-ordered backlog to change constantly, provided the latest version is used for sprint planning.

A set of software services evolves constantly, not unlike a living organism. New interfaces are released, whole services may split or merge, and individual services may go through significant redesign or deprecation. Ideally, teams within a company have a high level of autonomy and are "highly aligned, loosely coupled," to quote from the Netflix Culture document.[6]

By extension through Conway's law, the services operated by those teams should be independent. A lofty target is that any given team can implement 80% of the items in their backlog without needing any changes in the services on which they depend. Of the remaining 20%, a simple request to the owners of the remote service might result in a response indicating that the requested additional or altered interface makes sense, and it will be available by the time the requestor plans to start consuming it.

In the remaining cases the service owner may agree the requested change makes sense and fits with the service's roadmap, but its position on that roadmap is low compared with the priority that the requestor places on it. In such cases, the requestor might offer an "away team" to implement the requested change. An away team might consist of a pair of developers from the requestor team that temporarily joins the team that owns the service. The away team designs, tests, implements, and releases the requested change, all with stage-by-stage approval by the service owners who will be the long-term owners of the changes; when they are done they return to their "home team." A side effect of this away-team approach is cross-pollination of best practices, which can be particularly fruitful in a world where otherwise there is minimal communication between teams.

Agility. In an agile approach to product development, it's possible to establish a healthy balance between course correcting and optimizing for speed.

Even in a world of rapidly evolving requirements, it's OK for a team's well-ordered backlog to change constantly, provided the latest version is used for sprint planning. The team makes an explicit commitment to a set of tasks from the backlog and in return gets an uninterruptible window of protected time, a *sprint,* in which to work with as much speed as possible. Following the conclusion of this blissfully uninterrupted and churn-free period, the sprint demonstration shows the commitments that the team met.

Before the cycle continues with the next sprint-planning meeting using a course-corrected product backlog, the team holds a retrospective. This is an introspective session in which the team assesses the velocity reached and identifies ways to increase velocity in subsequent sprints. An honest retrospective, grounded in trust and self-awareness, can be used to figure out how to "sharpen the saw" before moving on to the next sprint.

Focus. Focus is necessary for achieving high velocity.

While Round was dreaming of a time when his team might be able to deploy their software to the Amazon website independently in less than a minute without needing to gain the approval of or even to notify anyone else, Andy Jassy was working on a vision document for a new business that would serve the needs of developers. In time, Jassy's AWS (Amazon Web Services) vision would coalesce around the need to help developers avoid "undifferentiated heavy lifting."

Teams want to focus on solving their customers' problems and on implementing, at high velocity, the business logic that is uniquely their responsibility. The heavy lifting of procuring, provisioning, and operating data centers, servers, and networks is a burden that they would rather not bear. They also want to avoid if at all possible being blocked by any people or processes they don't control (that is, that lie outside of their own team). As Bezos put it, "Even well-meaning gatekeepers slow innovation."[2] Cloud computing is an enabler for permissionless innovation and for moving toward software architectures that have a marked absence of gatekeepers and in which gatekeeping controls, such as access controls and compliance assertions, are programmatically enforced.

Culture. A high-velocity team pays attention to fostering a culture that encourages the team's talent to flourish and deliver results. This is self-reinforcing: teams with a culture that enables high velocity tend to disproportionately attract top talent. It's important to start with the presumption that people are talented, aligned with the mission, and want to work at high velocity. Some aspects of culture that positively impact velocity include diversity and inclusion, humility, trust, openness to learning, willingness to move with "urgency and focus,"[7] ownership, autonomy, and willingness to collectively commit to delivering results.

Enablement. To achieve high velocity, it's necessary to invest in systems that enable engineers to work at speed and to maximize the percentage of their time spent working on their area of unique responsibility. The obvious starting points are the tools and processes that they use to build, integrate, and deploy their code, and those used to operate their code after it has been released to ensure that it meets its requirements for availability, reliability, performance, and security.

Less obvious is the need to enable observability; while a services-based architecture may bring the benefits of autonomy and velocity, failures across service boundaries can be much more difficult to troubleshoot. It's helpful if metrics collection and propagation, monitoring, alarming, and issue tracking are common across services. Observability capabilities should enable distributed tracing, facilitating the precise detection of critical signals and indicators, and the progressive refinement of the search space, leading to pinpointing the root cause.

Experimentation. In the race to increase the rate at which they innovate, many companies actively seek to reduce the cost of running experiments so that they can do more of them. A higher rate of experimentation can facilitate more frequent course correcting. It's worth noting that a high rate of experimentation can be viewed as a high volume of discarded ideas, dead code, and failures.

Successful teams embrace failures, knowing that their models may be incomplete and that most of the incorrect choices they make are easily reversible. Ed Catmull, cofounder of Pixar, said, "Failure, when approached properly, can be an opportunity for growth. But the way most people interpret this assertion is that mistakes are a necessary evil. Mistakes aren't a necessary evil. They aren't evil at all. They are an inevitable consequence of doing something new and, as such, should be seen as valuable; without them, we'd have no originality."[4]

Conclusion. Software engineering occupies an increasingly critical role in companies across all sectors, but too many software initiatives end up both off target and over budget. A persistent myth is that effective delivery involves a perfect vision of what is needed combined with a plodding and unblinking march toward that vision, blind to all distractions or new information. A surer path is optimized for speed, open to experimentation and learning, agile, and subject to regular course correcting. C

Related articles on queue.acm.org

A Conversation with Werner Vogels
Jim Gray
https://queue.acm.org/detail.cfm?id=1142065

Conversations with Technology Leaders: Erik Meijer
Kate Matsudaira
https://queue.acm.org/detail.cfm?id=3092954

Meet the Virts
Tom Killalea
https://queue.acm.org/detail.cfm?id=1348589

References

1. Arkko, J. Permissionless innovation. IETF; https://www.ietf.org/blog/2013/05/permissionless-innovation/.
2. Bezos, J. Annual letter to Amazon shareholders, 2012.
3. Brooks Jr., F. *The Mythical Man-Month.* Addison-Wesley, 1975, 1995.
4. Catmull, E. *Creativity Inc.* Random House, 2014.
5. Killalea, T. The hidden dividends of microservices, *acmqueue 14*, 3 (2016); https://queue.acm.org/detail.cfm?id=2956643.
6. Netflix Culture. Netflix Jobs; https://jobs.netflix.com/culture.
7. Stripe. A quick guide to Stripe's culture; https://stripe.com/us/jobs/candidate-info?a=1#culture.
8. Vogels, W. Working backwards. All Things Distributed (Nov. 1, 2006); https://www.allthingsdistributed.com/2006/11/working_backwards.html.

Tom Killalea was with Amazon for 16 years and now provides advice to technology-driven companies and sits on the boards of Akamai, Capital One, Carbon Black, and MongoDB.

DOI:10.1145/3343046

Article development led by acmqueue
queue.acm.org

A discussion between Shaul Kfir and Camille Fournier.

DAML: The Contract Language of Distributed Ledgers

WHEN SHAUL KFIR cofounded Digital Asset in 2014, he was out to prove something to the financial services industry. He saw it as being not only hamstrung by an inefficient system for transaction reconciliation, but also in danger of missing out on what blockchain technology could do to address its shortcomings.

Since then, Digital Asset has gone to market with its own distributed-ledger technology, DAML (Digital Asset Modeling Language). And that does indeed take advantage of blockchain—only not in quite the way Kfir had initially intended. He and Digital Asset ended up taking an engineering "journey" to get to where they are today.

Kfir readily admits his own background in cryptography and cryptocurrency—both as a researcher (at Technion and MIT) and as a cryptocurrency entrepreneur in Israel—had more than just a little to do with the course that was originally charted. As for lessons learned along the way, Camille Fournier, the head of platform development for a leading New York City hedge fund, helps to elicit those here. She brings to the exercise her own background in distributed-systems consensus (as one of the original committers to the Apache Zookeeper Project) and financial services (as a former VP of technology at Goldman Sachs).

CAMILLE FOURNIER: One of the proposed applications of blockchain for the finance world now focuses on how it might be used to solve the distributed-ledger problem. How would you say that challenge is generally viewed at this point?

SHAUL KFIR: Let's start by considering the current state of IT in the financial industry. As any new product is introduced to this highly interconnected ecosystem, each organization builds a system to handle the new product's workflows. For each financial institution, this will become only one of many similar systems—so similar, in fact, they duplicate many of the same functions. Each of these systems, in turn, is integrated into other internal systems that all touch the same accounts, the same assets, and the same reference data. Beyond that, each organization is also faced with building reconciliation processes with each of its counterparties.

This already presents two major problems. First, if there are n new systems in the industry—one for each organization—you're soon going to end up with something on the order of n^2 new bilateral reconciliation relationships between the various entities. Second, you end up with a new system built for each new product instead of mutualizing the existing infrastructure and making it possible for products to be built and composed on top of each other.

As the financial industry or any product in it continues to mature, the natural tendency is to centralize in a

IMAGE FROM SHUTTERSTOCK.COM

hub-and-spoke manner around established central infrastructure providers so as to revert to something more like *n* different relationships, where everyone can just focus on reconciling against the infrastructure providers at the center of the network.

Another thing to bear in mind here is the way these systems are built typically provides only for reconciliation to happen in batch mode as an aggregate of all the transactions that have taken place over the course of the day. This is a historic remnant of how the financial system was originally designed and falls well short of the demands of today's financial world if only because there's too much intra-day risk associated with waiting until the end of the day to know with certainty what assets and liabilities you have.

FOURNIER: You're saying the current model relies too much on centralized reconciliation providers and increases intra-day risk by waiting until the end of the day to do reconciliation. How is a distributed ledger for transaction reconciliation going to address both of those concerns?

KFIR: Actually, there is a third concern I need to mention—and this is the most critical one to solve. As this ecosystem of distributed workflows continues to grow in complexity, it becomes more and more difficult to drive innovation, since each change forces each entity to make adjustments to its own internal systems and processes. The pace of change, as a consequence, eventually grinds to a halt.

Try to imagine a distributed ledger as a virtual SQL database the whole industry has access to—filtered, of course, such that each company can see only what it's allowed to see. It's with this mental model that distributed ledgers manage to address these problems in two main ways. One is that they reconcile the different companies' ledgers on a per-transaction basis in real time such that one shared database can show each of those companies exactly where it stands at any point. This means you don't have to wait until the end of the day to confirm your position.

The second major advantage is that a distributed ledger transforms the rollout of new workflows into a lightweight process. Imagine a company suggesting a new workflow simply by adding new tables and stored procedures to this database—which is to say, by doing something that resembles what SaaS [software as a service] companies currently do to provide for a continuous rollout of features. This will reduce friction and, as a consequence, lead to a faster pace of innovation, as existing workflows become composable elements of new workflows.

To clarify, think about how large technology companies use their infrastructure to achieve greater agility. Most of them today have some logically centralized infrastructure that includes a central code repository and a CI/CD [continuous integration/continuous delivery] system. If these ideas can be expanded to an industry level in the sense that you can start rolling out workflows as smart contracts that are written only once and then made available for everyone to build upon, that's clearly more efficient than leaving it to each organization to write its own workflows.

FOURNIER: Fine, but now let's back up for a second to talk about what actually distinguishes a distributed ledger from the classic reconciliation approach these large institutions at the center of the financial network are currently following.

KFIR: As things stand, these large institutions hold the ledgers of record for many different markets. There are at least two problems with that. One is that the systems used by many of these infrastructure providers are old—which is to say you can't use an API to go in and get a real-time view but instead are forced to wait for that night's batch to be run. Distributed ledgers don't address this problem directly, but they have been designed explicitly to take advantage of modern technology.

Then there's a second issue here that's more fundamental. A large financial institution cannot be in the position of needing to trust an API call into someone else's infrastructure to learn that something has just shifted on one of its trades by a few million dollars. Each financial institution holds its own ledger of record that it can make reference to whenever needed. But then, with at least one entity on either side of a trade, this means there will be at least two ledgers in play, both of which are going to require reconciliation around each event since they both touch the same data. A distributed ledger synchronizes these separate ledgers as if they were part of the same database while, in fact, each is controlled by one of the institutions that's party to the trade.

FOURNIER: So, this isn't just a problem with outdated technology. At least theoretically, you're saying a centralized reconciliation provider ought to be able to take advantage of newer hardware to become faster, more API driven, and less batch driven. But it seems the counterparties don't really trust the centralized third party to hold all this information on their behalf. Instead, what they really want is to have their own copy of that data, no matter what.

KFIR: Right ... because they need to verify, as well as trust.

FOURNIER: How does a distributed ledger differ from a centralized reconciliation system?

KFIR: First off, a distributed ledger is, of course, distributed. Each institution has a local ledger that is specific to that institution's assets. In this sense, the ledger is distributed so that each entity is able to view those shards of data that are relevant to it. At the same time, the integrity of the system provides the assurance that—if entity A, entity B, and entity C all have some specific data—their views of that data and the positions they hold with respect to it are consistent.

Another distinction is that most of the distributed ledgers today incorporate blockchains. But, over time, I think we will see more non-blockchain distributed ledgers that still manage to address these goals of consistency and infrastructure mutualization. What blockchain brings to the equation is complete trust—that is, how does each entity know that its view of the ledger represents the truth, the whole truth, and nothing but the truth?

Getting the truth from an engineering perspective is easy enough. A hash is a commitment to the veracity of some data.

As for "nothing but the truth," there are signatures to authenticate that the data is correct.

But then there is "the whole truth." This is where the blockchain structure comes in, with the append-only hash chain providing one means for ensuring that.

It isn't the *only* way to accomplish this, however. We need to distinguish between the technical meaning of blockchain—which refers to an immutable data structure that maintains an append-only chain of changes—and the more popular industry usage of blockchain, which can refer to anything having to do with distributed ledgers or, for that matter, anything that might be used to track the history of some distributed service. In fact, that's why we have come to use the term *distributed ledger technology*—or DLT—to imply this broader view.

FOURNIER: You have been careful to point out the blockchain data structure is only one of the options available. Is there some reason to consider other options?

KFIR: Actually, there is a problem with blockchain. Say we have two different companies that decide they want to do a swap, where one company lends one asset and gets some other asset in return. Each asset happens to be managed by a different ledger. One way to deal with this would be to use two-phase commits in much the same way cryptocurrencies employ hash locks. But this really puts the onus on the application developer or on whatever middleware you might happen to have in place. An alternative would be to ensure that the distributed ledgers are composable and able, by design, to merge and split seamlessly. There are a number of approaches for accomplishing this.

Conceptually, a blockchain design is just a single chain of hashes and so it doesn't really lend itself nicely to network composability. If you want to use a blockchain without sacrificing network composability, you need to take a completely different approach. I think it's critical for a distributed-ledger network to have this sort of composability.

FOURNIER: Before diving too deeply into the blockchain side of this, I'd like to know which parts of the centralized reconciliation problem proved to be especially challenging for you to address using this distributed-ledger approach.

KFIR: First, I should provide a bit of context. For one thing, it's important to acknowledge that consistency, data privacy, and liveness are all issues that become quite nuanced in any distributed system. Then there are a few things that are more specific to distributed ledgers. We have the challenge of ensuring confidentiality in the sense that people should be allowed to see only that slice of data that pertains directly to them, while also being assured that the integrity of the overall data model will be maintained. That's a nuanced computer science problem that involves the use of formal analysis of the protocols.

The second thing—and I don't think this problem is widely acknowledged in the industry—has to do with what I call the independence of SLAs [service-level agreements], or *liveness* in computer science terms. This means that, in a multiparty workflow, you don't want system downtime for a single party to hold up the overall process. For example, once an exchange matches a buyer and a seller—known as an *execution*—that's legally binding. If we were trying to employ the model here that most blockchains use, the exchange would report the trade by coordinating a transaction with a multiparty signature workflow. A minimum of three different parties would have to sign to acknowledge the trade: the exchange, the buyer, and the seller. And, in practice, there are always more than three parties. In fact, you have at least seven different parties that need to sign each trade execution transaction among all the different clearing participants, custodians, and so on.

Still, from a legal standpoint, if one of these parties' systems happens to be down or unresponsive, you don't want to hold back the whole workflow, since the trade is legally binding and must be entered into the ledger. Maintaining liveness—while also making sure that, if something needs to go through, it goes through—is a different problem to solve. To deal with that, we essentially had to build a very fine-grained delegation model similar to the OCAP [object capability] model.

Among the more significant lessons Kfir and his team at Digital Asset learned as they were striving to build a new system for transaction reconciliation was that they actually were dealing with more than just a distributed system problem. Much more, it turns out.

To their surprise, they found that each of the financial institutions they encountered had gone about implementing the industry's specifications for transaction reconciliation in a somewhat different fashion. This resulted in values in end-of-the-day reconciliations that didn't necessarily match from one institution to the next.

That's how they learned they also had a major data-modeling problem on their hands.

FOURNIER: Tell me more about the role blockchain plays in your distributed ledger.

What blockchain brings to the equation is complete trust—that is, how does each entity know that its view of the ledger represents the truth, the whole truth, and nothing but the truth?

KFIR: We don't use a blockchain data structure per se. Blockchains at this point work mostly with tokens. Whether it's Bitcoin or Ethereum or whatever, they employ tokens that are inherent first-class citizens. But that model quickly starts to break down as you add even a small amount of complexity.

The problem is that tokens are the digital equivalent of a bearer instrument in the financial world. They are simply unable to capture granular rights. For example, with a stock, the owner has the right to transfer, while the share registry has the right to split or merge. Cryptocurrencies and similar tokens don't capture these sorts of rights.

And then things become even more complex with corporate actions like voting rights. This is how we learned we would need to come up with an abstraction layer that isn't token-based. Which proved to be quite challenging. With that said, one of the Bitcoin elements we did end up keeping is its state-management model where each transaction consumes contracts and then creates new ones.

People often ask me, "Why did you feel the need to build a new contract language or a new abstraction layer?" I tell them we basically were forced to look for a new paradigm.

Just as the REST [representational state transfer] API didn't exist prior to 2000 for the simple reason that nobody really needed a RESTful architecture until the web came along, we found ourselves in a similar situation. We were tackling a new sort of problem—a workflow problem—and found we needed to come up with a new language and new abstraction layer. And we learned not to be afraid of doing that.

Interestingly, driven by customer demand, we've ported our language over to traditional SQL databases, because people believe that offers a powerful abstraction for modeling assets and workflows even in situations that don't really require the distributed aspect.

FOURNIER: OK, but I'm still wondering what initially made you think about blockchain as a potential solution to your reconciliation problem. What suggested blockchain would be an obvious fit?

KFIR: It was just my personal naivete. We have other people in the company who came from other disciplines, but my background is in cryptography and cryptocurrency, so I was still drinking the blockchain Kool-Aid in 2012. This led me to think, "These bank IT people probably just don't know what they're doing, so I'll show them how blockchain can make things better."

FOURNIER: That sounds somewhat familiar. I'd say you were not alone in drinking that Kool-Aid.

KFIR: As we started to dig in, we spoke with people in many different roles at a number of financial institutions and started to see some troubling patterns emerge. One was that the pace of innovation throughout the industry was sluggish at best. That's what first led us to think, "OK, maybe blockchain actually *does* make sense here, given that a smart contract language could really help these people roll out something faster."

But that's also when we really shifted our focus to a more holistic view: "How can we make this industry more productive and agile?" Which completely shifted our value proposition. Our mission today is to accelerate the pace of innovation in these industries by building a powerful abstraction, along with a powerful set of developer tools.

Another factor that really surprised us at first had to do with the problem of garbage-in, garbage-out data. Basically, as we were trying to figure out what we should do, we found that the data structuring and message structuring across virtually every asset proved to be just horrible. Everyone seemed to have a different view of how the data ought to look and, while most of the industry standards spelled out specifications for messages between participants, they didn't address the semantics. So, everyone had implemented the specifications for transaction reconciliation differently. There were all these variances from the low level on up. This led to end-of-day reconciliations where the values didn't actually match from one institution to the next.

Once we saw that, we knew this wasn't just a distributed-systems problem, but also an issue that involved the abstractions and data structures. That's when we started to approach things in an entirely different manner. As a company, we realized we were tackling a problem that was bigger than we at the time were equipped to handle.

It's important to acknowledge that consistency, data privacy, and liveness are all issues that become quite nuanced in any distributed system.

Once we accepted that, we joined forces with another company, Zurich-based Elevence, which we later ended up acquiring. Elevence was made up primarily of programming-language experts, formal-methods experts, and ex-quants who had been working on the very same data-modeling problem we had been puzzling over, and they had come to realize their approach was one that would work particularly well on a distributed ledger. We, on the other hand, felt pretty good about our distributed-ledger work, but we knew we needed help with the data modeling. As it turns out, we were both quite fortunate to find each other.

FOURNIER: So, you put your distributed ledger together with the data-modeling work from Elevence to more or less solve your system-design problems. But what did that leave you with from all that blockchain work you had initially done on your distributed ledger? What portion of that work has proven to remain stable and at least somewhat useful to you?

KFIR: The most important work that remains from our early days is a robust abstract model of a distributed ledger. That's to say, in the course of designing the interface between an ideal domain language and a practical distributed ledger, we had to design a formally specified abstract model of a blockchain that included a data-structuring model, an integrity model, and a privacy model.

While we initially built all this for our own internal needs, it proved robust enough to let us later integrate DAML with numerous other distributed ledgers, including not only our own but also others from VMware, Hyperledger, and more. This also allowed us to have a common language to use in presenting to users the tradeoffs between different DAML platforms.

FOURNIER: Did you encounter any other problems that you think are going to cause some real grief in the blockchain world?

KFIR: There are situations where it's going to prove difficult to get the blockchain tokens to match up. Tokens simply aren't expressive enough to represent any application that presents even moderate complexity. That tends to get lost in all the blockchain hyperbole.

There's also a much bigger problem we needed to deal with—though this doesn't appear to have been much of an issue on many other platforms to date—and that is sub-transaction privacy. Even within a single transaction, different entities have different rights as to what they can see. For example, if you and I were to swap a share for cash, our banks should see only a cash transfer, while the share registry should see only the share transfer—even while the two of us are able to see the whole transaction. If there's some other blockchain that's able to accomplish this, we're not aware of it.

Another big revelation for Kfir and his colleagues came with the realization that the problem they were addressing was less a technical one than the general burden of ever-growing complexity. Which is to say that, rather than building custom solutions the institutions themselves could then deploy, the focus should instead be on creating tools that programmers at each of the institutions could use to create their own solutions.

With this, the emphasis shifted to creating a synchronization layer to provide for consistency among all the trading partners, and complementing that with a contract language and an abstraction layer with enough support for system functionality to let developers at each institution focus more on business logic than on more traditional programming concerns.

FOURNIER: You mentioned your decision to partner with—and ultimately acquire—Elevence in order to take advantage of its expertise with formal methods and programming languages. What drove you to make that decision, and do you think the absence of just such expertise may have had something to do with the failures that have occurred over the years in the distributed-ledger space?

KFIR: I wouldn't be so quick to label those previous efforts as failures. I just think of them as things that happened back in the early days of distributed ledgers. Remember the first of these projects started up in the 1980s, well before the Internet boom. In the same way, I don't think we failed initially, either. We moved forward in one direction before recognizing that our initial model, which bore a lot of similarities to cryptocurrencies, was one that many people would have difficulty working with as complexity grew over time. Basically, you would end up needing a Ph.D. in cryptography or security to write your distributed-ledger applications. Which obviously wouldn't get us very close to achieving our goal of increasing productivity and accelerating innovation.

We started out with a model that was very token-based but then realized we needed to be more workflow-centric as we faced growing complexity. In effect, we moved to a model in which you have different action types for the different sorts of things you need to do, rather like Ripple transaction types. [Ripple is a financial-transaction protocol based on a shared public ledger.] This has worked well in Ripple's case since that's focused on a very specific domain—namely, payments. It could put the emphasis on defining certain types of transactions as the sorts of things you're allowed to do with your system.

As opposed to Ripple, though, we weren't focused on a particular domain but rather on building a flexible SDK for a number of domains. For each of those we had separate classes of transactors called *actions*. But then, in the course of doing code reviews, we kept finding bugs, which led us to conclude we were putting too much trust in the ability of our SDK users to write secure actions and deal with signatures and things of that nature.

That's when it dawned on us that we needed to stop resorting to custom solutions each time we ran across a problem and instead turn our attention to coming up with a common programming remedy.

At first, we thought we would write an abstraction layer and develop a programming language to go along with it. But then we found that the team at Elevence had already developed a top-down approach to address just such issues. So, at root, we knew we had a problem, and they clearly had the solution. It was just that obvious.

FOURNIER: What does your architecture look like now that you've put it all together?

KFIR: There are a number of components, but let's start with what we call our Abstract Ledger Model, which is an implementable specification for running DAML applications on all kinds of ledgers—by which I mean distributed ledgers and traditional databases and graph databases and event streams and even things we haven't thought up yet. Obviously, we have our own distributed-ledger platform, but that's also complemented by implementations for other persistence layers like PostgreSQL and VMware.

We also have what we call the DAML engine. Whereas DAML is our contract language, which you can think of as being built to provide for safety in multiparty workflows, the DAML engine is what provides for the interpretation of contracts written in that language. It also ensures that the authorized commands that act on the current state of the distributed ledger result in a new state that's consistent with the contract semantics written in DAML. All of these layers sit within what we call our ledger server, which is a swappable component that implements the Abstract Ledger Model along with a runtime environment for DAML programs.

Other ledger providers now have implemented the same DAML abstraction and APIs using an integration kit that we've open sourced. This allows users to choose different blockchains depending on their own requirements and vendor preferences.

On top of all that is an application and integration framework that employs an event-handler model. This means that applications can react to events from the ledger while also submitting commands to it. Basically, that framework provides all the properties required to support caching, system restarts, optimizations, and the like. It attends to these sorts of issues so the developer can focus on writing the business logic. This is similar to the experience of using the serverless or lambda functions available from AWS

(Amazon Web Services), GCP (Google Cloud Platform), and Microsoft Azure.

FOURNIER: That's impressive! Anything else?

KFIR: We also have an SDK, which isn't part of the platform architecture itself, but it does come with an embedded distributed-ledger simulator, as well as a testing language that provides a full semantic model of a distributed ledger—meaning that if it will run on your local machine, it will run on the platform as well. That includes analysis tools that leverage the structure of the language to check for common mistakes and entitlement logic. From the contracts you write, it derives who, from among all the different participants on the network, should be entitled to see this particular data. All the entitlement logic is driven by the workflows, so that you, the developer, don't have to specify any of that yourself.

FOURNIER: How did you manage that?

KFIR: That actually comes from the structure of the language. As you specify your business logic, you specify the workflows. Then, with reference to the function signature, you can see all the network participants that will be affected by this contract or might come to be affected later on. This then allows you to analyze that for all future evolutions of the contract.

FOURNIER: Cool! It sounds like you ended up with something that could prove to be quite powerful. Now tell me about some of the more interesting engineering lessons you learned along the way.

KFIR: From an engineering-management perspective, there has been a lot for me to learn. I came to this from being a cryptographer who was working on cutting-edge zero-knowledge proofs and SNARKs (succinct non-interactive arguments of knowledge), so my intent was just to keep on using all those cool tools when we founded the company. Well, just as the token model fell short of our expectations, cutting-edge crypto also wasn't such a great fit. That was one lesson.

Another lesson is that, while I'd never written in a functional programming language before, I'm now a complete convert to both functional programming and strongly typed languages. I've come to believe that the industry as a whole needs to evolve toward defensive programming techniques. Functional programming and strong types are among the best tools available for that.

I also learned that you can never be wedded to the tools you have or the code you've built. A year after launching this company, we had already completely scrapped all the code we had written and left behind most of the tools we had been using. Maybe we should have noticed earlier that we could manage all states on the ledger itself, but the important thing is that we ultimately *did* discover that. And that was largely because we listened to our customers to find out what their real problems were. This is how we found we were going in the wrong direction. Mind you, we didn't do *exactly* what the customers told us we should do to resolve those problems, since that would have put us completely off track as well.

I've also learned that you need to put careful consideration into every single trade-off. I'll give you an example: Immutability of a blockchain makes resolution and recovery from data corruption a nontrivial problem. Accenture came out with a paper promoting mutable blockchains and was ridiculed for it. Having come from the blockchain community, I understand the reasons for the ridicule, but the paper itself was actually pretty balanced. It's just that the blockchain true believers weren't really capable of reading it in an unbiased way. I've learned you can't afford to be doctrinaire like that, since you need to be free to take an analytical approach to every single significant tradeoff that comes your way.

FOURNIER: If this model you've built for distributed ledgers and the formal methods you use to define them prove to be widely adopted, what sorts of changes do you expect will follow? I ask since one of the things you said you hope to accomplish is to open up innovation throughout the industry.

KFIR: I think we will see the same kind of Cambrian explosion we witnessed in the web world once we started using mutualized infrastructure in public clouds and frameworks. An example I often cite is that I had absolutely no knowledge of web development when I started a Bitcoin brokerage in Israel some years ago, since I was very much a low-level assembler and crypto kind of guy at the time. But then it took only three weeks for me to learn enough Ruby on Rails and Heroku to push out the first version of a management system for that brokerage. And that's because I had to think only about the models, the views, and the controllers—which is to say only the business processes. The hardest part, of course, had to do with building a secure wallet, but that was my expertise.

Anyway, moving on to today's banking world, if we look at all the networks of financial institutions that currently have mutualized workflows, you won't find any frameworks, abstraction layers, or mutualization of infrastructure. There's nothing that people with good ideas can use to rapidly roll out new systems and then iterate on them or upgrade them. That means if someone comes up with financial innovations that might be used, for example, in the health-care domain to reduce payment counterparty risk, there's just no easy way right now to deploy that.

But should we find ourselves in a world where every large enterprise is part of one of these distributed networks that does mutualize infrastructure, then people would be able to iterate very quickly on these sorts of ideas. Basically, they would be able to propose new products and deploy them as smart contracts. The impact of that could prove especially significant in some of the large industries that so far have been slow to adopt innovations. I think the potential there could be huge. C

Related articles on queue.acm.org

Bitcoin's Academic Pedigree
Arvind Narayanan and Jeremy Clark
https://queue.acm.org/detail.cfm?id=3136559

Research for Practice: Cryptocurrencies, Blockchains, and Smart Contracts
Arvind Narayanan and Andrew Miller
https://queue.acm.org/detail.cfm?id=3043967

Bitcoin's Underlying Incentives
Yonatan Sompolinsky and Aviv Zohar
https://queue.acm.org/detail.cfm?id=3168362

Attention: Undergraduate *and* Graduate Computing Students

There's an ACM Student Research Competition (SRC) at a SIG Conference of interest to you!

STUDENT RESEARCH COMPETITION

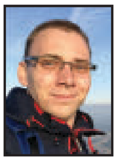
Association for Computing Machinery
Advancing Computing as a Science & Profession

SPONSORED BY Microsoft

It's hard to put the ACM Student Research Competition experience into words, but we'll try…

"Attending ACM SRC was a transformative experience for me. It was an opportunity to take my research to a new level, beyond the network of my home university. Most important, it was a chance to make new connections and encounter new ideas that had a lasting impact on my academic life. I can't recommend ACM SRC enough to any student who is looking to expand the horizons of their research endeavors."

David Mueller
North Carolina State University | SIGDOC 2018

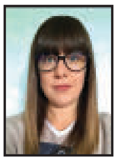


"The SRC was a great chance to present early results of my work to an international audience. Especially the feedback during the poster session helped me to steer my work in the right direction and gave me a huge motivation boost. Together with the connections and friendships I made, I found the SRC to be a positive experience."

Matthias Springer
Tokyo Institute of Technology | SPLASH 2018

"At the ACM SRC, I got to learn about the work done in a variety of different research areas and experience the energy and enthusiasm of everyone involved. I was extremely inspired by my fellow competitors and was happy to discover better ways of explaining my own work to others. I would like to specifically encourage undergraduate students to not hesitate and apply! Thank you to all those who make this competition possible for students like me."

Elizaveta Tremsina
UC Berkeley | TAPIA 2018



"Joining the Student Research Competition of ACM gave me the opportunity to measure my skills as a researcher and to carry out a preliminary study by myself. Moreover, I believe that "healthy competition" is always challenging in order to improve yourself. I suggest that every Ph.D. student try this experience."

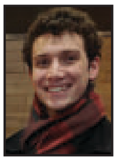
Gemma Catolino
University of Salerno | MobileSoft 2018

"Participating in the ACM SRC was a unique opportunity for practicing my presentation skills, getting feedback on my work, and networking with both leading researchers and fellow SRC participants. Winning the competition was a great honor, a motivation to continue working in research, and a useful boost for my career. I highly recommend any aspiring student researcher to participate in the SRC."

Manuel Rigger
Johannes Kepler University Linz, Austria | Programming 2018

"I have been a part of many conferences before both as an author and as a volunteer but I found SRC to be an incredible conference experience. It gave me the opportunity to have the most immersive experience, improving my skills as a presenter, researcher, and scientist. Over the several phases of ACM SRC, I had the opportunity to present my work both formally (as a research talk and research paper) and informally (in poster or demonstration session). Having talked to a diverse range of researchers, I believe my work has much broader visibility now and I was able to get deep insights and feedback on my future projects. ACM SRC played a critical role in facilitating my research, giving me the most productive conference experience."

Muhammad Ali Gulzar
University of California, Los Angeles | ICSE 2018



"The ACM SRC was an incredible opportunity for me to present my research to a wide audience of experts. I received invaluable, supportive feedback about my research and presentation style, and I am sure that the lessons I learned from the experience will stay with me for the rest of my career as a researcher. Participating in the SRC has also made me feel much more comfortable speaking to other researchers in my field, both about my work as well as projects I am not involved in. I would strongly recommend students interested in research to apply to an ACM SRC—there's really no reason not to!"

Justin Lubin
University of Chicago | SPLASH 2018

Check the SRC Submission Dates: ***https://src.acm.org/submissions***

- Participants receive: $500 (USD) travel expenses
- All Winners receive a medal and monetary award. First place winners advance to the SRC Grand Finals
- Grand Finals Winners receive a handsome certificate and monetary award at the ACM Awards Banquet

Questions? Contact Nanette Hernandez, ACM's SRC Coordinator: ***hernandez@hq.acm.org***

DOI:10.1145/3339399

CARLA GOMES
Cornell University
THOMAS DIETTERICH
Oregon State University
CHRISTOPHER BARRETT
Cornell University
JON CONRAD
Cornell University
BISTRA DILKINA
University of Southern California
STEFANO ERMON
Stanford University
FEI FANG
Carnegie Mellon University
ANDREW FARNSWORTH
Cornell University
ALAN FERN
Oregon State University
XIAOLI FERN
Oregon State University
DANIEL FINK
Cornell University
DOUGLAS FISHER
Vanderbilt University
ALEXANDER FLECKER
Cornell University
DANIEL FREUND
Massachusetts Institute of Technology
ANGELA FULLER
U.S. Geological Survey
JOHN GREGOIRE
California Institute of Technology
JOHN HOPCROFT
Cornell University
STEVE KELLING
Cornell University
ZICO KOLTER
Carnegie Mellon University
WARREN POWELL
Princeton University
NICOLE SINTOV
The Ohio State University
JOHN SELKER
Oregon State University
BART SELMAN
Cornell University
DANIEL SHELDON
University of Massachusetts Amherst
DAVID SHMOYS
Cornell University
MILIND TAMBE
Harvard University
WENG-KEEN WONG
Oregon State University
CHRISTOPHER WOOD
Cornell University
XIAOJIAN WU
Microsoft AI Research
YEXIANG XUE
Purdue University
AMULYA YADAV
Pennsylvania State University
ABDUL-AZIZ YAKUBU
Howard University
MARY LOU ZEEMAN
Bowdoin College

Computer and information scientists join forces with other fields to help solve societal and environmental challenges facing humanity, in pursuit of a sustainable future.

Computational Sustainability: Computing for a Better World and a Sustainable Future

THESE ARE EXCITING times for computational sciences with the digital revolution permeating a variety of areas and radically transforming business, science, and our daily lives. The Internet and the World Wide Web, GPS, satellite communications, remote sensing, and smartphones are dramatically accelerating the pace of discovery, engendering globally connected networks of people and devices. The rise of practically relevant artificial intelligence (AI) is also playing an increasing part in this revolution, fostering e-commerce, social networks, personalized medicine, IBM Watson and AlphaGo, self-driving cars, and other groundbreaking transformations.

Unfortunately, humanity is also facing tremendous challenges. Nearly a billion people still live below the international poverty line and human activities and climate change are threatening our planet and the livelihood of current and future generations. Moreover, the impact of computing and information

9 INDUSTRY, INNOVATION AND INFRASTRUCTURE
17 PARTNERSHIPS FOR THE GOALS
1 NO POVERTY

IMAGE COMPOSITION BY ANDRIJ BORYS ASSOCIATES; PHOTO BY MARIJA STOJKOVIC

technology has been uneven, mainly benefiting profitable sectors, with fewer societal and environmental benefits, further exacerbating inequalities and the destruction of our planet.

Our vision is that computer scientists can and should play a key role in helping address societal and environmental challenges in pursuit of a sustainable future, while also advancing computer science as a discipline.

For over a decade, we have been deeply engaged in computational research to address societal and environmental challenges, while nurturing the new field of *Computational Sustainability*. Computational sustainability aims to identify, formalize, and provide solutions to computational problems concerning the balancing of environmental, economic, and societal needs for a sustainable future.[18] Sustainability problems offer challenges but also opportunities for the advancement of the state of the art of computing and information science. While in recent years increasingly more computer and information scientists have engaged in research efforts focused on social good and sustainability,[12,14,16,24,29,31,35] such computational expertise is far from the critical mass required to address the formidable societal and sustainability challenges that we face today. We hope our work in computational sustainability will inspire more computational scientists to pursue initiatives of broad societal impact.

Toward a Sustainable Future

In 1987, *Our Common Future*, a United Nations report by the World Commission on Environment and Development,[a] raised serious concerns about the state of our planet, the livelihood of current and future generations, and introduced the groundbreaking notion of "sustainable development."

Sustainable development is development that meets the needs of the present without compromising the ability of future generations to meet their needs.

a United Nations. *Our Common Future.* Retrieved Aug. 25, 2018; http://www.un-documents.net/our-common-future.pdf

» key insights

- **Computer science enriches sustainability. Computer scientists can and should make important contributions to help address key societal and environmental challenges facing humanity, in pursuit of a sustainable future. The new field of computational sustainability brings these efforts together.**
- **Sustainability enriches computer science. In turn, working on sustainability problems, which involve uncertainty, machine learning, optimization, remote sensing, and decision making, enriches computer science by generating compelling new computational challenge problems.**
- **Sustainability concerns human well-being and the protection of the planet. A large group of computer science researchers, collaborating with an even larger group of domain specialists from social, environmental, and natural sciences, can drive computational sustainability in ways that would not be possible in a smaller or less interdisciplinary setting.**

The sustainable development goals (SDGs)[b] identify areas of critical importance for human well-being and the protection of the planet and seek to integrate and balance the economic, social, and environment dimensions for sustainable development (see Figure 1).[c]

Computational Research in Sustainability

We illustrate some of our computational sustainability research, which has focused on three general sustainability themes: Balancing environmental and socioeconomic needs; biodiversity and conservation; and, renewable and sustainable energy and materials. The research is also centered on three broad computational themes: optimization, dynamical models, and simulation; data and machine learning; and, multi-agent systems, crowdsourcing, and citizen science (noted in Figure 2). This section is organized in terms of our three sustainability themes, highlighting crosscutting computational themes, as depicted in the subway lines of Figure 3.

Balancing environmental and socioeconomic needs. The elimination of poverty is one of the most challenging sustainable development goals. Globally, over 800 million people live below the international poverty line of $1.90 per person per day.[d] Rapid population growth, ecosystem conversion, and new threats due to conflicts and climate change are further pushing several regions into chronic poverty.

The lack of reliable data is a major obstacle to the implementation of policies concerning poverty, food security, and disaster relief. In particular, policies to eradicate poverty require the ability to identify who the poor are and where they live. Poverty mapping can be very challenging, especially in the case of the developing countries, which typically suffer from large deficiencies in terms of data quantity, quality, and analysis capabilities. For example, some countries have not collected census data in decades.[e] To mitigate this challenge, Ermon and collaborators are introducing novel approaches for obtaining large-scale spatial and temporal socioeconomic indicators from publicly available satellite and remote sensing data (Figure 4). The approaches take advantage of advances in machine learning and are quite effective for estimating a variety of socio-economic indicators of poverty, even comparable to the predictive performance of expensive survey data collected in the field, and are currently being used by the World Bank.[20]

In the arid regions of sub-Saharan Africa, one of the world's poorest regions, migratory pastoralists manage and herd livestock as their primary occupation. During dry seasons they must migrate from their home villages to remote pastures and water points. Barrett and collaborators are developing models for studying well-being dynamics and poverty traps associated with migratory herders and other populations.[5] The herders' preferences are also key in the design of policies for sustainable development. Unfortunately, such preferences are often unknown to policymakers and must be inferred from data. Ermon et al.[11] developed generative models based on (inverse) reinforcement learning and dynamic discrete choice models, to infer the spatiotemporal preferences of migratory pastoralists, which provide key information to policymakers concerning a variety of decisions, in particular, the locations for adding new watering points for the herders.

Access to insurance is critical since uninsured losses can lead to a vicious cycle of poverty.[5,8] Unfortunately, agricultural and disaster insurance are either unavailable or prohibitively expensive in many developing countries, due to the lack of weather data and other services. To mitigate this problem, the Trans-Africa Hydro-Meteorological Observatory (TAHMO) project is designing and deploying a network of 20,000 low-cost weather stations throughout sub-Saharan Africa.[36] This project gives rise to challenging stochastic optimization and learning problems for optimal weather station site selection and for uncertainty quantification in the sensors and weather predictions. For example, precipitation, one of the most important variables for agriculture, is challenging to predict due to its heavy-tailed nature and the malfunctions of rain gauges. Dietterich and his collaborators are developing models for detecting instrument malfunctions and also conditional mixture models to capture the high variance of the phenomena. There are other

b United Nations. *Transforming Our World: The 2030 Agenda for Sustainable Development.* Retrieved Aug. 25, 2018; http://www.un.org/ga/search/view_doc.asp?symbol=A/RES/70/&Lang=E

c United Nations. *The Sustainable Development Goals Report.* Retrieved Aug. 25, 2018; https://bit.ly/2WbeKNB.

d We used 2013 data since the 2015 survey data coverage is too low and considerable Asia data are suppressed; http://iresearch.worldbank.org/PovcalNet/povDuplicateWB.aspx

e United Nations. *A World That Counts: Mobilizing the Data Revolution for Sustainable Development.* Retrieved June 16, 2018; http://www.undatarevolution.org/wp-content/uploads/2014/12/A-World-That-Counts2.pdf

Figure 1. On Sept. 25, 2015, under the auspices of the United Nations and as part of a wider 2030 Agenda for Sustainable Development, 193 countries agreed on a set of 17 ambitious goals, referred to as the Sustainable Development Goals (SDGs), to end poverty, protect the planet, and ensure prosperity for all.

SUSTAINABLE DEVELOPMENT GOALS
1 NO POVERTY
2 ZERO HUNGER
3 GOOD HEALTH AND WELL-BEING
4 QUALITY EDUCATION
5 GENDER EQUALITY
6 CLEAN WATER AND SANITATION
7 AFFORDABLE AND CLEAN ENERGY
8 DECENT WORK AND ECONOMIC GROWTH
9 INDUSTRY, INNOVATION AND INFRASTRUCTURE
10 REDUCED INEQUALITIES
11 SUSTAINABLE CITIES AND COMMUNITIES
12 RESPONSIBLE CONSUMPTION AND PRODUCTION
13 CLIMATE ACTION
14 LIFE BELOW WATER
15 LIFE ON LAND
16 PEACE, JUSTICE AND STRONG INSTITUTIONS
17 PARTNERSHIPS FOR THE GOALS
SUSTAINABLE DEVELOPMENT GOALS

challenges in agriculture, in particular, due to market failures and information asymmetries—a consistent problem in environmental policy.[8,23]

There are also many challenges and opportunities in connection with social interventions in the U.S., where more than 40 million people live below the U.S. poverty threshold. The U.S. also has the highest infant mortality rate and the highest youth poverty rate in the Organization for Economic Cooperation and Development, which comprises 37 high-income economies regarded as the developed countries.[f] For example, Los Angeles County has over 5,000 youth between the ages of 13 and 24 sleeping on the streets or living in emergency shelters on any given day. In the context of homeless youth drop-in centers in Los Angeles, Yadav et al.[40] propose novel influence maximization algorithms for peer-led HIV prevention, illustrating how AI algorithms can significantly improve dissemination of HIV prevention information among homeless youth and have real impact on the lives of homeless youth. Tambe and Rice[35] provide a compilation of other examples of AI for social work concerning HIV prevention, substance abuse prevention, suicide prevention, and other social work topics.

As a final example on balancing environmental and socioeconomic issues, consider the urban landscape, which is far more congested than it was 10, 20, or 50 years ago. There is a critical need to provide individualized transportation options that have smaller carbon footprints than the automobile. One emerging alternative is bike-sharing which allows for multimodal commute round trips, with a great degree of individual flexibility, as well as economic, environmental, and health benefits. These systems have given rise to a host of challenging logistical problems, whose computationally efficient solution is required to make this new alternative sustainable. The algorithmic requirements for these problems bring together issues from discrete optimization, stochastic modeling, and behavioral eco-

f Statement on visit to the U.S. by Philip Alston, United Nations Special Rapporteur on extreme poverty and human rights (2018). Retrieved June 16, 2018; http://socialprotection-humanrights.org/wpcontent/uploads/2018/06/G1812530.pdf

Figure 2. Our research is focused on three general sustainability areas depicted as the faces of the triangle.

Solutions to problems in each of these areas draw on a combination of three broad computational themes, depicted as circles. Each sustainability application creates a transformative synthesis by incorporating a combination of computational techniques from any of these themes, while each computational technique is in turn applied to a variety of problems.

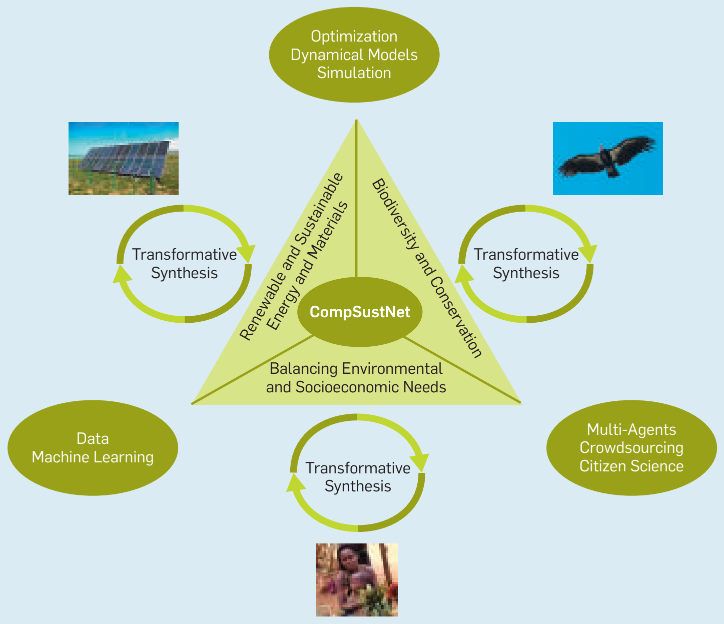


Figure 3. Subway lines highlight examples of general domain crosscutting computational problem classes identified in our research projects, which correspond to subway stops.

A junction, where one project appears on multiple lines, shows how one sustainability problem can bring together multiple computational problem classes. Similarly, each subway line shows how otherwise disparate sustainability applications are related through computational problem classes.

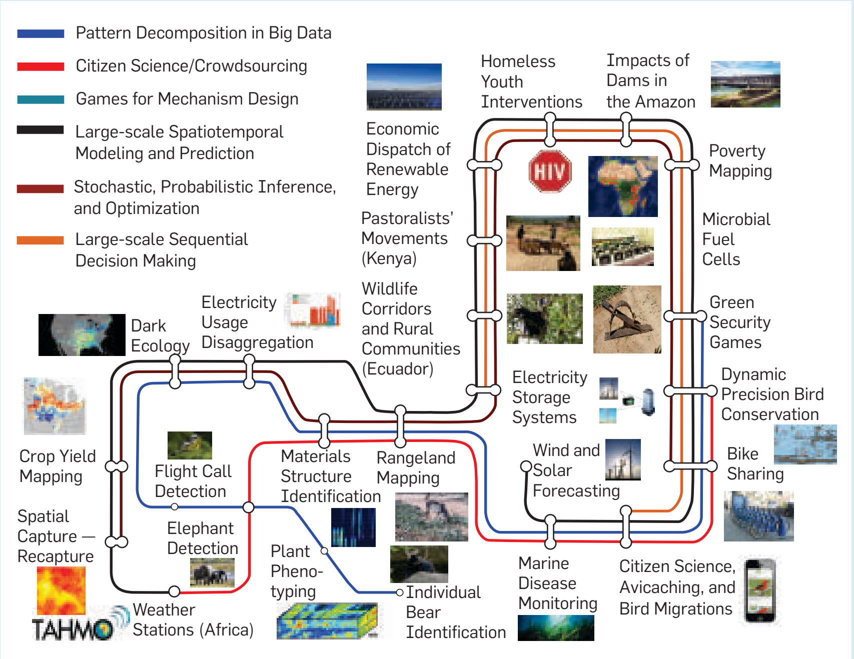

Pattern Decomposition in Big Data
Citizen Science/Crowdsourcing
Games for Mechanism Design
Large-scale Spatiotemporal Modeling and Prediction
Stochastic, Probabilistic Inference, and Optimization
Large-scale Sequential Decision Making
Homeless Youth Interventions
Impacts of Dams in the Amazon
Economic Dispatch of Renewable Energy
Poverty Mapping
Pastoralists' Movements (Kenya)
Microbial Fuel Cells
Wildlife Corridors and Rural Communities (Ecuador)
Green Security Games
Electricity Usage Disaggregation
Dark Ecology
Electricity Storage Systems
Dynamic Precision Bird Conservation
Crop Yield Mapping
Flight Call Detection
Materials Structure Identification
Rangeland Mapping
Wind and Solar Forecasting
Bike Sharing
Spatial Capture — Recapture
Elephant Detection
Plant Pheno-typing
Individual Bear Identification
Marine Disease Monitoring
Citizen Science, Avicaching, and Bird Migrations
TAHMO
Weather Stations (Africa)
HIV

Figure 4. Transfer learning is an effective approach to model and predict socioeconomic indicators in data-scarce regions that takes advantage of satellite images that are globally available, updated frequently, and becoming increasingly more accurate.

The approach first trains a deep convolutional neural network to predict nighttime light intensity (a good proxy for economic activity) based on daytime satellite imagery. The model then estimates average household expenditures based on expenditure data from the World Bank's Living Standards Measurement Study surveys. This is done via semi-supervised learning, while enforcing spatial consistency using a Gaussian process on top of the neural network. The resulting model is surprisingly accurate, explains close to 70% of the variation in the data in some countries, and outperforms all previous methods including methods based on proprietary phone meta-data (not publicly available). This general approach has been adapted for large-scale spatial and temporal modeling and prediction of a variety of socioeconomic indicators.[20]

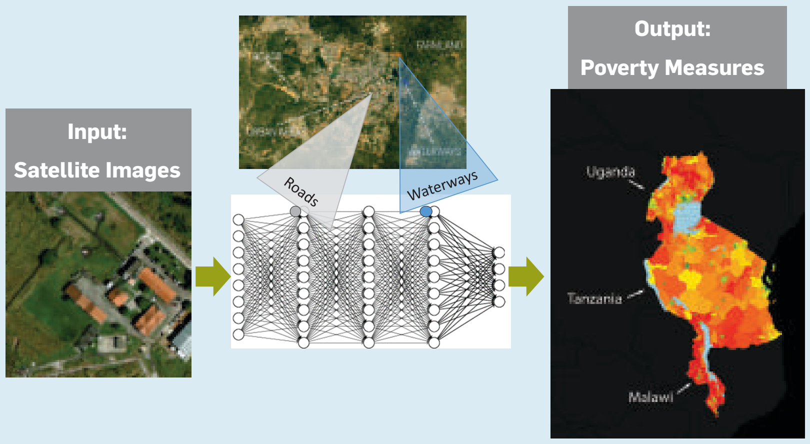
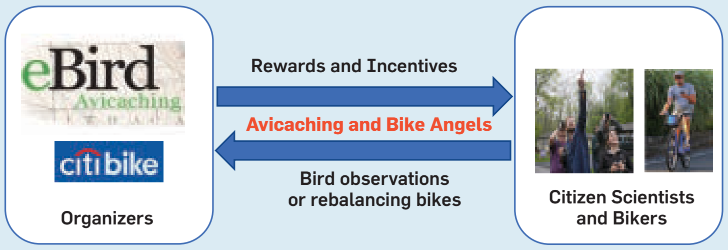


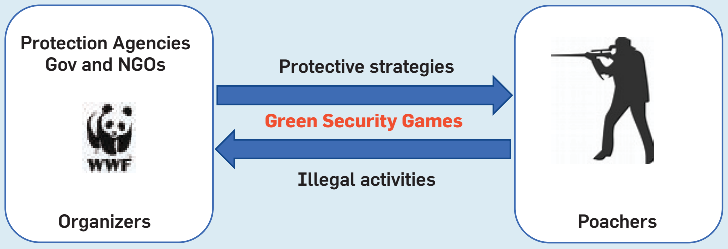
Figure 5. Games for mechanism design.

Games for mechanism design lead to challenging bi-level stochastic optimization and learning problems in which the game organizer must take into account the preferences of the agents (citizen scientists, bikers, or poachers) with respect to the organizer's actions, in order to identify the best incentive or protection strategy.


eBird Avicaching
citibike
Organizers
Rewards and Incentives
Avicaching and Bike Angels
Bird observations or rebalancing bikes
Citizen Scientists and Bikers

Games for bias reduction in observation collections and re-balancing activities

(a) The Avicaching game incentivizes citizen scientists to submit bird observations from undersampled areas;[39] Bike Angels incentivizes NYC bikers to rebalance Citi bikes.[17]



Games for combatting poaching and illegal activities

(b) Green security games strategically protect natural resources (forests, fish areas, and so on) against poaching and illegal activities.[13]

nomics, as well as mechanism design to appropriately incentivize desired collective behavior. One striking recent success is the crowdsourcing approach to rebalancing the shared bike fleet in NYC that contributes more to the effectiveness of Citi Bike than all motorized efforts (Figure 5); this and other computational challenges in this emerging domain are surveyed by Freund et al.[17]

Biodiversity and conservation. Accelerated biodiversity loss is another great challenge threatening our planet and humanity, especially considering the growing evidence of the importance of biodiversity for sustaining ecosystem services. The current rate of species extinction is estimated to be 100 to 1,000 times the background rates that were typical over Earth's history. Agriculture, urbanization, and deforestation are main causes of biodiversity reduction, leading to habitat loss and fragmentation. Climate change and introduction by humans of species to non-native ecosystems are further accelerating biodiversity loss.[28]

A fundamental question in biodiversity research and conservation concerns understanding how different species are distributed across landscapes over time, which gives rise to challenging large-scale spatial and temporal modeling and prediction problems.[15,25] Species distribution modeling is highly complex as we are interested in simultaneously predicting the distribution of hundreds of species, rather than a single species at a time, as traditionally done due to computational challenges. Motivated by this problem, Chen et al.[7] developed the Deep Multivariate Probit Model (DMVP), an end-to-end learning approach for the multivariate probit model (MVP), which captures interactions of *any multi-entity process*, assuming an underlying Gaussian distribution[7] (Figure 6), and scales considerably better than previous approaches.

Citizen science programs play a key role in conservation efforts and, in particular, in providing observational data. eBird, a citizen science program of the Cornell Lab of Ornithology, has over 450,000 members, who have gathered more than 650 million bird observations, corresponding to over 30,000,000 hours of field work.[34] Furthermore, to complement eBird observational data, other information sources are exploited. For example, Sheldon and collaborators' Dark Ecology project[33] extracts biological information from weather data. eBird data, combined with large volumes of environmental data and our spatiotemporal statistical and machine learning models of bird species occurrence and abundance, provide habitat preferences of the birds at a fine resolution, leading to novel approaches for bird conservation.[27] The results from the eBird species distribution models formed the basis for the 2011–2017 U.S. Department of Interior's State of the Birds (SOTB) reports.

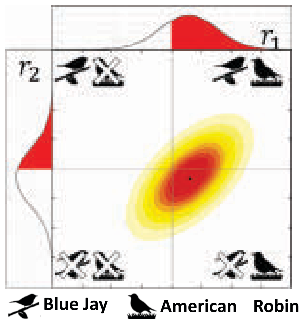
The SOTB reports are generating tremendous interest from conservation organizations in using species distribution results to improve bird conservation. A good example is The Nature Conservancy's *Bird Returns* program.[27] The program uses reverse combinatorial auctions, in which farmers are compensated for creating habitat conditions for birds by keeping water in their rice fields for the periods that coincide with bird migrations. This novel market-based approach is only possible given the fine-grained bird habitat preference provided by the eBird-based models. Bird Returns has been tremendously successful and has created thousand of additional acres of habitat for migratory birds.

Other challenges concerning quantification and visualization of uncertainty in species prediction, multiscale data fusion and interpretation from multiple sensors, incorporation of biological and ecological constraints, and models of migration have also been addressed.[30,32–34] Sheldon and collaborators introduced collective graphical models, which can model a variety of aggregate phenomena, even though they were originally motivated for modeling bird migrations[6,32] (Figure 7).

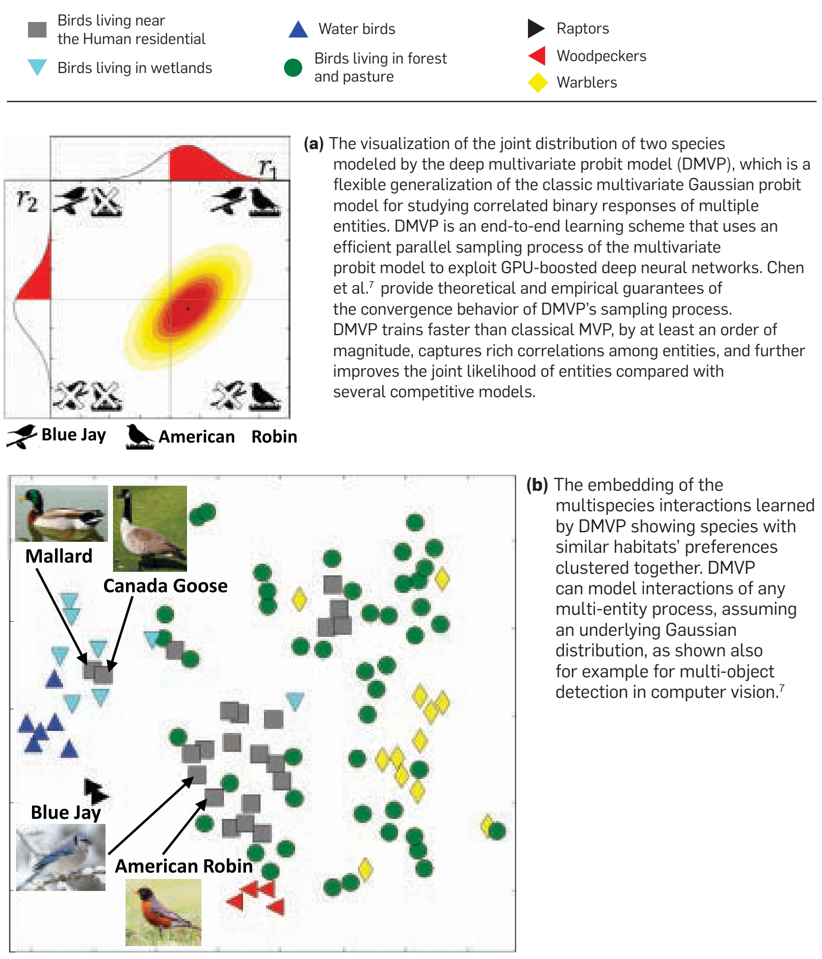
Citizen science, while a valuable source of information for species distribution modeling, also poses several computational challenges with respect to imperfect detection, variable expertise in citizen scientists,[21] and spatial and temporal sampling bias.[34,39] The Avicaching game was developed to combat sample bias in eBird submissions (Figure 5).

To mitigate the various habitat threats encountered by species, several conservation actions are adopted. For example,

Figure 6. Multi-entity interactions.



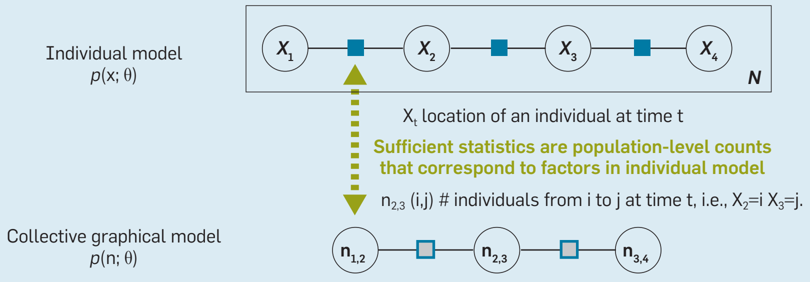
(a) The visualization of the joint distribution of two species modeled by the deep multivariate probit model (DMVP), which is a flexible generalization of the classic multivariate Gaussian probit model for studying correlated binary responses of multiple entities. DMVP is an end-to-end learning scheme that uses an efficient parallel sampling process of the multivariate probit model to exploit GPU-boosted deep neural networks. Chen et al.[7] provide theoretical and empirical guarantees of the convergence behavior of DMVP's sampling process. DMVP trains faster than classical MVP, by at least an order of magnitude, captures rich correlations among entities, and further improves the joint likelihood of entities compared with several competitive models.



(b) The embedding of the multispecies interactions learned by DMVP showing species with similar habitats' preferences clustered together. DMVP can model interactions of any multi-entity process, assuming an underlying Gaussian distribution, as shown also for example for multi-object detection in computer vision.[7]

Figure 7. Collective graphical models (CGMs) are a general-purpose formalism for conducting probabilistic inference about a large population of individuals that are only observed in aggregate.

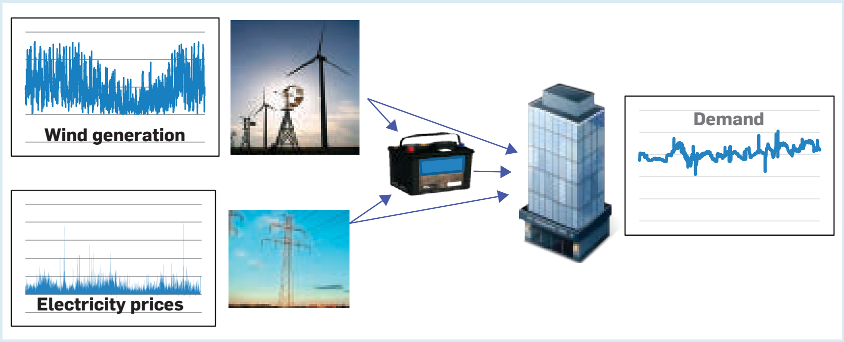
The generality of CGMs makes them suitable to model a range of aggregate phenomena, from bird migrations (the initial motivating application), to differential privacy.[6,32] Formally, CGMs are a probabilistic model for the sufficient statistics of a graphical model, for which incomplete and noisy observations are available. Sheldon and collaborators have contributed a number of inference and learning algorithms and theoretical results about CGMs with surprising connections to the theory of belief propagation, and fast message-passing algorithms based on the Bethe entropy. The figure depicts a high-level representation of a collective graphical model. Noisy and incomplete observations **y** (not shown) are made of the sufficient statistics through a noise model $p(y \mid n)$, and the goals are to perform inference by computing the posterior distribution $p(n \mid y)$ and to learn the parameters θ of the individual model.



Individual model $p(x; \theta)$
X_1
X_2
X_3
X_4
N
X_t location of an individual at time t
Sufficient statistics are population-level counts that correspond to factors in individual model
$n_{2,3}$ (i,j) # individuals from i to j at time t, i.e., X_2=i X_3=j.
Collective graphical model $p(n; \theta)$
$n_{1,2}$
$n_{2,3}$
$n_{3,4}$

Figure 8. Robust planning of an efficient energy system to serve a load (building) from a wind farm (with variable wind speeds), the grid (with variable prices), and a battery storage device is challenging.

Energy storage provides a smooth, dispatchable flow of energy, matching energy when it is generated to loads when they arise. Motivated by his work in energy and other applications, Powell[26] proposes a unified modeling framework for sequential decision making, covering several distinct fields that deal with (sequential) decisions and uncertainty (dynamic programming, stochastic programming, stochastic control, simulation optimization, and bandit problems, among others) under a common umbrella. In this unified framework, there are four fundamental classes of policies consisting of policy function approximations (PFAs), cost function approximations (CFAs), policies based on value function approximations (VFAs), and look-ahead policies.



wildlife corridors have been shown to be an effective way to combat habitat fragmentation. The design of wildlife corridors, typically under tight conservation budgets, gives rise to challenging stochastic optimization problems. Current approaches to connect core conservation areas through corridors typically consider the movement of a *single* species at a time. Dilkina et al.[9] propose new computational approaches for optimizing corridors considering benefit-cost and trade-off analysis for landscape connectivity conservation for multispecies. The results demonstrate economies of scale and complementarities conservation planners can achieve by optimizing corridor designs for financial costs and multiple species jointly. Another related work integrates spatial capture-recapture models into reserve design optimization. In yet another related effort, Fuller and collaborators are developing a program focused on Ecuador's Choco-Andean Biological Corridor, which comprises two of the world's most significant biodiversity hotspots, that integrates landscape connectivity for Andean bears and other species with economic, social and ecological information.

Prevention of wildlife crime is also important in conservation. In recent years there has been considerable AI research on devising wildlife monitoring strategies and simultaneously providing rangers with decision aids. The approaches use AI to better understand patterns in wildlife poaching and enhance security to combat poaching (for example, see Fang et al.[14]). This work is leading to research advances at the intersection of computational and behavioral game theory and data-driven optimization. A notable example of this research developed so-called *green security games* (Figure 5) and has led to an application tool named Protection Assistant for Wildlife Security (PAWS),[13] which has been tested and deployed in several countries, including Malaysia, Uganda, Botswana, and China.

Finally, we mention non-native invasive species, which invade both land and water systems and threaten ecosystems' ability to house biodiversity and provide ecosystem services. For example, the invasion of tamarisk trees in the Rio Grande valley in New Mexico has greatly reduced the amount of water available for native species and for irrigation of agricultural crops. Bio-economic models provide a basis for policy optimization and sensitivity analysis, by capturing the complex dynamics of the ecosystem, that is, the processes by which the invasive species is introduced to the landscape and spreads, as well the costs and effects of the available management actions. Unfortunately, often not much is known about these processes. Albers et al.[2] demonstrate the power of a stylized simulator-defined Markov decision process approach for tamarisk, using a complex dynamical bio-economic model. A challenge is to scale up the approach and increase the realism of the bio-economic models.

Renewable and sustainable energy and materials. Renewables are being integrated into the smart grid in ever increasing amounts. Because renewables like wind and solar are non-dispatchable resources, they cannot be scheduled in advance, and alternative generation methods have to be scheduled to make up the difference. The variability and uncertainty of renewables have also raised the importance of energy storage (Figure 8). However, storage is expensive, and different storage technologies and settings are required to meet needs such as frequency regulation, energy shifting, peak shifting, and backup power. In general, controlling energy systems (generation, transmission, storage, investment) involves a number of new challenging learning and optimization problems.

For example, SMART-Invest[22] is a stochastic dynamic planning model, which is capable of optimizing investment decisions in different electricity generation technologies. SMART-Invest consists of two layers. The first is an outer optimization layer that applies stochastic search to optimize investments in wind, solar, and storage. The objective function is non-convex, non-smooth, and only available via an expensive-to-evaluate black box function. The approach exploits approximate convexity to solve this optimization quickly and reliably. The second layer captures hourly variations of wind and solar over an entire year, with detailed modeling of day-ahead commitments, forecast uncertainties and ramping constraints. SMART-Invest produces a more realistic picture of an optimal mix of wind, solar, and storage than previous approaches, and therefore can provide more accurate guidance for policy makers.

In another example concerning the placement of hydropower dams in the Amazon basin (Figure 9), Wu et al.[38] propose new exact and approximation multi-objective optimization approaches, which are key to simultaneously consider different sustainability criteria.

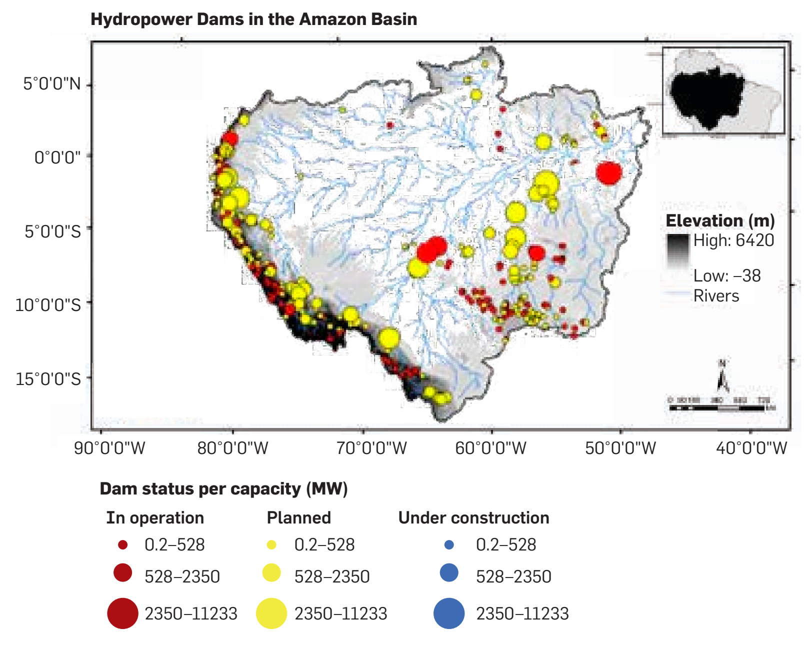
In yet another example, Donti et al.[10] propose task-based model learning, which was inspired by scheduling electricity generation. Task-based model learning is a general approach that combines data learning and decision making (for example, a stochastic optimization problem) in an end-to-end learning framework, specifying a loss function in terms of the decision-making objective. In this approach all components are differentiable, and therefore it is possible to learn the model parameters to improve the closed-loop performance of the overall system, which is a novel way to train machine learning models based upon the performance of decision-making systems.

Finally, we highlight new sustainable materials and processes. They provide a fundamental basis for solutions to some of the most pressing issues in energy, as well as more general issues in sustainability. In many cases, long-term solutions will depend on breakthrough innovations in materials, such as the development of new materials and processes for more efficient batteries, fuel cells, solar fuels, microbial fuel cells, or for CO_2 reduction. The high cost of conventional single-sample synthesis and analysis are driving the scientific communities to explore so-called high-throughput experimentation to accelerate the discovery process. This setup leads to computational challenges for designing and planning the experiments. Furthermore, the data analysis, integration, and interpretation process are key bottlenecks that are expert-labor intensive. Current state-of-the-art machine learning techniques are not able to produce physically meaningful solutions. Efficient computational methods are therefore urgently needed for analyzing the flood of high-throughput data to obtain scientific insights. A promising research direction is the development of generative models for unsupervised learning and for providing supervision using domain knowledge through theory-based models and simulators.

As an example, in high-throughput materials discovery, a challenging problem is the so-called phase-map identification problem, an inverse problem in which one would like to infer the crystal structures of the materials deposited onto a thin film based on the X-ray diffraction (XRD) patterns of sample points. This problem can be viewed as topic modeling or source separation with intricate physics constraints since the observed diffraction pattern of a sample point may consist of a mixture of several pure crystal patterns, and some of them may not be sampled. The task is further complicated by the inherent noise in the measurements. Human experts analyze the diffraction patterns by taking into account knowledge of the underlying physics and chemistry of materials, but it is a very labor-intensive task and often it is very challenging even for human experts. This is a good example that completely defies the current state of the art of machine learning. Phase-Mapper,[4] is an AI platform that tightly integrates results from XRD experimentation with learning, reasoning, and human insights, to infer crystal structures from XRD data. In particular, Phase-Mapper integrates an efficient relaxed projection method for constrained non-negative matrix factorization that incorporates physics constraints; prior knowledge based on known patterns from inorganic crystal structure databases, as well as human computation strategies. In addition Phase-Mapper uses theory-based models for incorporating prior knowledge. Since the deployment of Phase-Mapper at Caltech's Joint Center for Artificial Photosynthesis, thousands of XRD patterns have been processed, resulting in the discovery of new energy materials, such as a new family of metal oxide solar light absorbers. Gomes, Gregoire and collaborators are developing SARA (Scientific Autonomous Robotic Agent; http://bit.ly/2M8efm9) that encapsulates the scientific method for accelerating materials discovery substantially extending Phase Mapper. Finally, we point out a related source separation problem—hyper spectral plant phenotyping—that is

Figure 9. Multi-objective learning and optimization.

In many sustainability problems, it is critical to jointly consider multiple, often conflicting, objectives. This is the case for hydropower dam planning in the Amazon basin, with about 350 new hydropower dams proposed, which will dramatically affect a variety of Amazon ecosystem services, such as biodiversity, sediment transport, freshwater fisheries, navigation, besides energy production. The Pareto frontier captures the trade-offs between the multiple objectives with respect to the different non-dominated solutions. The non-dominated solutions also provide valuable information concerning the dams' ranking. We have developed exact dynamic-programming algorithms, fully polynomial time approximation schemes (FPTAS), and other approaches for computing the Pareto frontier for tree-structured networks, with application to the Amazon hydropower dam placement problem. For example, we can now approximate the Pareto frontier for the entire Amazon basin ($\sim$ 4M river segments), with respect to three criteria (energy; river connectivity, a good proxy for fish migrations and navigation; and sediment production) within 5% from the true optimal Pareto frontier in about 4 minutes (8 threads); or within 2% in about 1.2 hours (8 threads). The results, combined with visualization tools, help policymakers make more informed decisions concerning multiple criteria and different spatial scales.[38]



tackled in Wahabzada[37] with probabilistic topic models.

Another area that can benefit dramatically from advanced AI and machine learning methods is the planning and design of scientific experiments. For example, Fern and collaborators are developing novel machine learning and constraint budgeted optimization techniques to help scientists design more efficient experiments for microbial fuels by allowing them to efficiently explore different nano-structures.[3] They employ Bayesian optimization with resource constraints and production actions and have developed a new general Monte Carlo tree search algorithm with theoretical guarantees. This work also led to a large-scale empirical evaluation of Bayesian optimization algorithms, which was motivated by the confusing landscape of results in Bayesian optimization. The study involved implementing a number of top algorithms within a common framework and using cloud resources to run comparisons on a large number and variety of test functions. The main result of the study was to show the well-known Bayesian optimization heuristic—*expected improvement*—performed as well as any other approach in general and often won by significant margins. This includes beating methods such as the arguably more popular upper confidence bound (UCB) algorithm. The study found that algorithms such as UCB, which require setting a parameter for controlling exploration, are very sensitive to the parameters, making them difficult to apply widely. Expected improvement is parameter-free and appears to be quite robust. Krause and collaborators also apply Bayesian optimization for maximum power point tracking in photovoltaic power plants.[1]

As a final example, Grover et al.[19] model the search for the best charging policy for the Li-ion battery chemistry as a stochastic multi-arm bandit with delayed feedback. They found policies (functions for making decisions based on state variables) that considerably outperform current policies (by up to 35% in experimentation time).

Computational Synergies

We have highlighted how computational sustainability problems encompass a combination of distinguishing aspects that make them unique in scale, impact, complexity, and richness, posing new challenges and opportunities for computing and information science, leading to transformative research directions. One of our key goals has been to identify classes of computational problems that cut across a variety of sustainability (and other) domains. Given the universality of computational thinking, findings in one domain can be transferred to other domains. Examples of high-level crosscutting computational problem classes, some of them depicted in Figure 3, include *spatiotemporal modeling and prediction* for bird conservation, poverty mapping, and weather mapping; *sequential decision making* for managing (renewable) resources, designing scientific experiments, managing invasive species, and pastoralism interventions; *pattern decomposition with complex constraints* for phase map identification in materials discovery, identification of elephant and bird calls from audio recordings, inferring plant phenotypes from hyper spectral data and scientific topic modeling; *active learning* (not shown in Figure 3), for scientific experimentation and sensor placement, including citizen science, and crowdsourcing, and *games for mechanism design* for providing incentives for citizen scientists, placing patrols and drones to combat poaching and illegal fishing, or incentivizing bikers to balance bike stations.

Citizen science programs play a key role in conservation efforts, particularly in providing observational data. eBird, a citizen science program of the Cornell Lab of Ornithology, has over 450,000 members, who have gathered more than 650 million bird observations.

We believe that pursuing research in core or paradigmatic crosscutting computational problems is a sine qua non condition to ensure the cohesiveness and growth of computational sustainability as a field, so that researchers develop general models and algorithms with application in different sustainability and other domains. Our experience shows these *core problems naturally emerge out of real-world sustainability-driven projects, approached with the perspective of lifting solution methods to produce general methodologies, as opposed to only solving narrow problem scenarios.*

In this article, we focused on computational sustainability research examples from CompSustNet,[g] a computational sustainability research network involving a large number of researchers. Unfortunately, we are not able to

g http://www.compsust.net/

include many other exciting research contributions and computational challenges raised by sustainability questions, as identified in computer science, engineering, and social and natural sciences. Examples include the role of large-scale distributed systems and sensor networks, the Internet of Things, cyber-physical systems, cyber security, privacy, fairness, accountability, transparency for advanced computational systems, and also fundamental computational concepts such as reliability, modeling the hierarchical structure of socio-technical systems, and human-in-the-loop systems and intuitive, user-friendly interfaces. We also only touched on some of the 17 U.N. sustainable development goals. We point the reader to the increasing number of conferences and journals that are now starting to include tracks, workshops, and special issues focusing on tackling sustainability and societal issues, bringing together different computing and information science areas (HCI, systems, AI, and algorithms, among others).

Planning for a sustainable future encompasses complex interdisciplinary decisions for balancing environmental, economic, and societal needs, which involve significant computational challenges, requiring expertise and research efforts in computing and information science and related disciplines. Computational sustainability aims to develop new computational methodologies to help address such environmental, economic, and societal challenges. The continued dramatic advances in digital platforms, computer software and hardware, sensor networks and the Internet of Things continue to provide significant new opportunities for accelerating the pace of discovery to address societal and sustainability issues. Computational sustainability is a two-way street: it injects computational ideas, thinking, and methodologies into addressing sustainability questions but it also leads to foundational contributions to computing and information science by exposing computer scientists to new challenging problems, formalisms, and concepts from other disciplines. Just as sustainability issues intersect an ever-increasing cross-section of emerging scientific application domains, computational sustainability broadens the scope and diversity of computing and information science while having profound societal impact.

Acknowledgments. We thank the CompSustNet members for their many contributions to computational sustainability and the support of two NSF Expeditions in Computing awards (CNS-0832782 and CCF-1522054). We thank the anonymous reviewers for their suggestions to improve the manuscript. Any use of trade, firm, or product names is for descriptive purposes only and does not imply endorsement by the U.S. government.

References

1. Abdelrahman, H., Berkenkamp, F., Poland, J., and Krause, A. Bayesian optimization for maximum power point tracking in photovoltaic power plants. In *Proceedings of the 2016 European Control Con.* (Aalborg, Denmark, June 29–July 1, 2016), 2078–2083. https://doi.org/10.1109/ECC.2016.7810598
2. Albers, J.H., Dietterich, T., Hall, K., Katherine, L., and Taleghan, M. Simulator-defined Markov decision processes: A case study in managing bio-invasions. *Artificial Intelligence and Conservation* (2nd. ed.). F. Fang, M. Tambe, B. Dilkina, and A. Plumptre, (Eds.). Cambridge Univ. Press, 2018
3. Azimi, J., Fern, X., and Fern, A. Budgeted optimization with constrained experiments. *J. Artif. Intell. Res. 56* (2016), 119–152.
4. Bai, J. et al. Phase mapper: Accelerating materials discovery with AI. *AI Mag. 39*, 1 (2018), 15–26.
5. Barrett, C., Garg, T., and McBride, L. Well-being dynamics and poverty traps. *Annual Review of Resource Economics 8* (2016), 303–327.
6. Bernstein, G., McKenna, R., Sun, T., Sheldon, D., Hay, M., and Miklau, G. Differentially private learning of undirected graphical models using collective graphical models. In *Proceedings of the 34th International Conference on Machine Learning*, 2017, 478–487.
7. Chen, D., Xue, Y., and Gomes, C. End-to-end learning for the deep multivariate probit model. *ICML* (2018).
8. Coble, K., Mishra, A., Ferrell, S., and Griffin, T. Big data in agriculture: A challenge for the future. *Applied Economic Perspectives and Policy 40*, 1 (2018), 79–96. https://doi.org/10.1093/aepp/ppx056
9. Dilkina, B. et al. Trade-offs and efficiencies in optimal budget-constrained multi-species corridor networks. *Conservation Biology 31*, 1 (2017), 192–202.
10. Donti, P., Kolter, J.Z., and Amos, B. Task-based end-to-end model learning in stochastic optimization. In *Proceedings of Advances in Neural Information Processing Systemsg Systems* (Long Beach, CA, USA, Dec. 4–9, 2017), 5490–5500.
11. Ermon, S. et al. Learning large-scale dynamic discrete choice models of spatio-temporal preferences with application to migratory pastoralism in East Africa. In *AAAI*, 2015, 644–650.
12. Faghmous, J. and Kumar, V. A big data guide to understanding climate change: The case for theory-guided data science. *Big Data 2*, 3 (2014), 155–163.
13. Fang, F. et al. PAWS—A deployed game-theoretic application to combat poaching. *AI Magazine 38*, 1 (2017), 23–36.
14. Fang, F., Tambe, M., Dilkina, B., and Plumptre, A (eds.). *Artificial Intelligence and Conservation.* Cambridge University Press, 2018.
15. Fink, D. et al. Spatiotemporal exploratory models for broad-scale survey data. *Ecological Applications 20*, 8 (2010), 2131–2147.
16. Fisher, D.H. Recent advances in AI for computational sustainability. *IEEE Intelligent Systems 31*, 4 (2016), 56–61; https://doi.org/10.1109/MIS.2016.61
17. Freund, D., Henderson, S.G., and Shmoys, D.B. *Sharing Economy: Making Supply Meet Demand.* Springer, 2018.
18. Gomes, C.P. Computational sustainability: Computational methods for a sustainable environment, economy, and society. *The Bridge 39*, 4 (2009), 5–13.
19. Grover, A. et al. Best arm identification in multi-armed bandits with delayed feedback. In *Proceedings of the Inter. Conf. Artificial Intelligence and Statistics* (Playa Blanca, Lanzarote, Canary Islands, Spain, April 9–11, 2018), 833–842.
20. Jean, N., Burke, M., Xie, M., Davis, W.M., Lobell, D.B., and Ermon, S. Combining satellite imagery and machine learning to predict poverty. *Science 353*, 6301 (2016), 790–794.
21. Kelling, S. et al. Can observation skills of citizen scientists be estimated using species accumulation curves? *PloS one 10*, 10 (2015).
22. Khazaei, J. and Powell, W.B. SMART-Invest: A stochastic, dynamic planning for optimizing investments in wind, solar, and storage in the presence of fossil fuels. The case of the PJM electricity market. *Energy Systems 9*, 2 (2018), 277–303.
23. Kraus, S. Automated negotiation and decision-making in multiagent environments. *ECCAI Advanced Course on Artificial Intelligence.* Springer, 2001,150–172.
24. Lässig, J., Kersting, K., and Morik, K (Eds.). Computational Sustainability. *Studies in Computational Intelligence 645* (2016). Springer.
25. Phillips, S.J., Anderson, R.P., and Schapire, R.E. Maximum entropy modeling of species geographic distributions. *Ecological Modeling 190*, 3-4 (2006), 231–259.
26. Powell, W. A unified framework for stochastic optimization. *European J. Operational Research 275*, 3 (2019), 795–821.
27. Reynolds, M.D. et al. Dynamic conservation for migratory species. *Science Advances 3*, 8 (2017), e1700707.
28. Rockström, J. et al. Planetary boundaries: Exploring the safe operating space for humanity. *Ecology and Society 14*, 2 (2009).
29. Rudin, C. and Wagstaff, K. Machine learning for science and society. *Machine Learning 95*, 1 (2014), 1–9; https://doi.org/10.1007/s10994- 013- 5425- 9
30. Ruiz-Muñoz, J.F., You, Z., Raich, R., and Fern, X.Z. Dictionary learning for bioacoustics monitoring with applications to species classification. *Signal Processing Systems 90*, 2 (2018), 233–247.
31. Russell, S.J. et al. Letter to the editor: Research priorities for robust and beneficial artificial intelligence: An open letter. *AI Magazine 36*, 4 (2015).
32. Sheldon, D.R. and Dietterich, T.G. Collective graphical models. *Advances in Neural Information Processing Systems*, 2011, 1161–1169.
33. Sheldon, D.R. et al. Approximate Bayesian inference for reconstructing velocities of migrating birds from weather radar. In *Proceedings of the 27th AAAI Conf. Artificial Intelligence.* (Bellevue, WA, USA, July 14-18, 2013).
34. Sullivan, B.L. et al. The eBird enterprise: An integrated approach to development and application of citizen science. *Biological Conservation 169* (2014), 31–40.
35. Tambe, M. and Rice, E (eds.). *Artificial Intelligence and Social Work.* Cambridge University Press, 2018.
36. Giesen, N., Hut, R., and Selker, J. The Trans-African Hydro-Meteorological Observatory (TAHMO). *Wiley Interdisciplinary Reviews: Water 1*, 4 (2014), 341–348.
37. Wahabzada, M., Mahlein, A.K., Bauckhage, C., Steiner, U., Oerke, E.C., and Kersting, K. Plant phenotyping using probabilistic topic models: Uncovering the hyperspectral language of plants. *Scientific Reports 6* (2016).
38. Wu, X. et al. Efficiently approximating the Pareto frontier: Hydropower dam placement in the Amazon basin. In *AAAI* (2018).
39. Xue, Y., Davies, I., Fink, D., Wood, C., and Gomes, C.P. Avicaching: A two stage game for bias reduction in citizen science. In *Proceedings of the 2016 Intl. Conf. on Autonomous Agents & Multiagent Systems.*776–785.
40. Yadav, A. et al. Influence maximization in the field: The arduous journey from emerging to deployed application. In *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems*, 2017, 150–158.

For questions regarding this article, contact **Carla Gomes** (gomes@cs.cornell.edu), a professor of computer science and director of the Institute for Computational Sustainability at Cornell University, Ithaca, NY, USA

Watch the authors discuss this work in the exclusive *Communications* video. https://cacm.acm.org/videos/computational-sustainability

DOI:10.1145/3338843

Exploiting a simple, expressive logic based on relations to describe designs and automate their analysis.

BY DANIEL JACKSON

Alloy: A Language and Tool for Exploring Software Designs

ALLOY IS A language and a toolkit for exploring the kinds of structures that arise in many software designs. This article aims to give readers a flavor of Alloy in action, and some examples of its applications to date, thus giving a sense of how it can be used in software design work.

Software involves structures of many sorts: architectures, database schemas, network topologies, ontologies, and so on. When designing a software system, you need to be able to express the structures essential to the design and to check that they have the properties you expect.

You can express a structure by sketching it on a napkin. That's a good start, but it's limited. Informal representations give inconsistent interpretations, and they cannot be analyzed mechanically. So people have turned to formal notations that define structure and behavior precisely and objectively, and that can exploit the power of computation.

By using formality early in development, you can minimize the costs of ambiguity and get feedback on your work by running analyses. The most popular approach to advocate this is agile development, in which the formal representation is code in a traditional programming language and the analysis is conventional unit testing.

As a language for exploring designs, however, code is imperfect. It's verbose and often indirect, and it does not allow *partial* descriptions in which some details are left to be resolved later. And testing, as a way to analyze designs, leaves much to be desired. It's notoriously incomplete and burdensome, since you need to write test cases explicitly. And it's very difficult to use code to articulate design without getting mired in low-level details (such as the choice of data representations).

An alternative, which has been explored since the 1970s, is to use a design language built not on conventional machine instructions but on logic. Partiality is free because rather than listing each step of a computation, you write a logical constraint saying what's true after, and that constraint can say as little or as much as you please. To analyze such a language, you use specialized algorithms such as model checkers or satisfiability solvers (more on these later). This usually requires much less effort than testing, since you

» key insights

- Using a simple logic of relations, Alloy lets you model software designs that involve complex, evolving structures.
- Alloy's tool uses SAT technology to simulate designs and find subtle flaws, and has been used in a wide variety of applications from networking and security to critical systems.
- A key advantage of logical modeling is that you can construct a design incrementally, in an agile way, representing and analyzing only an essential subset of the behavioral contraints.
- Alloy is complementary to a class of tools called model checkers, and is a valuable addition to the software designer's toolkit.

only need to express the property you want to check rather than a large collection of test cases. And the analysis is much more complete than testing, because it effectively covers all (or almost all) test cases that you could have written by hand.

What Came Before: Theorem Provers and Model Checkers

To understand Alloy, it helps to know a bit about the context in which it was developed and the tools that existed at the time.

Theorem provers are mechanical aids for constructing mathematical proofs. To apply a theorem prover to a software design problem, you formulate some intended property of the design, and then attempt to prove the theorem that the property follows from the design. Theorem provers tend to provide very rich logics, so they can usually express any property the designer might want, at least about states and state transitions—more dynamic properties can require a temporal logic that theorem provers don't typically support directly. Also, because they generate mathematical proofs, which can be checked by tools that are smaller and simpler than the tool that finds the proof, you can be confident the analysis is sound.

On the down side, the combination of an expressive logic and sound proof means that finding proofs cannot generally be automated. Theorem provers usually require considerable effort and expertise from the user, often orders of magnitude greater than the effort of constructing a formal design in the first place. Moreover, failure to find a proof does not mean that a proof does not exist, and theorem provers don't provide counterexamples that explain concretely why a theorem is not valid. So theorem provers are not so useful when the intended property does not hold—which unfortunately is the common case in design work.

Model checkers revolutionized design analysis by providing exactly the features theorem provers lacked. They offer push-button automation, requiring the user to give only the design and property to be checked. They allow dynamic properties to be expressed (through temporal logics), and generate counterexamples when properties do not hold. Model checkers work by exploring the space of possible states of a system, and if that space is large, they may require considerable computational resources (or may fail to terminate). The so-called "state explosion" problem arises because model checkers are often used to analyze designs involving components that run in parallel, resulting in an overall state space that grows exponentially with the number of components.

Alloy was inspired by the successes and limitations of model checkers. For designs involving parallelism and simple state (comprising Boolean variables, bounded integers, enumerations and fixed-size arrays), model checkers were ideal. They could easily find subtle synchronization bugs that appeared only in rare scenarios that involved long traces with multiple context switches, and therefore eluded testing.

For hardware designs, model checkers were often a good match. But for software designs they were less ideal. Although some software design problems involve this kind of synchronization, often the complexity arises from the structure of the state itself. Early model checkers (such as SMV[9]) had limited expressiveness in this regard, and did not support rich structures such as trees, lists, tables, and graphs.

Explicit state model checkers, such as SPIN,[14] and later Java Pathfinder,[37] allowed designs with rich state to be modeled, but, despite providing support for temporal properties, gave little help for expressing structural ones. To express reachability (for example, that two social media users are connected by some path of friend edges), you would typically need to code an explicit search, which would have to be executed at every point at which the property was needed. Also, explicit state model checkers have limited support for partiality (since the model checker would have to conduct a costly search through possible next states to find one satisfying the constraints).

Particularly difficult for all model checkers are the kinds of designs that involve a *configuration* of elements in a graph or tree structure. Many network protocols are designed to work irrespective of the initial configuration (or of the configuration as it evolves), and exposing a flaw often involves not only finding a behavior that breaks a property but also finding a configuration in which to execute it.

Even the few model checkers that can express rich structures are generally not up to this task. Enumerating possible configurations is not feasible, because the number of configurations grows super-exponentially: if there are n nodes, there are $2^{n \times n}$ ways to connect them.

Alloy's Innovations

Alloy offered a new kind of design language and analysis that was made possible by three innovations:

Relational logic. Alloy uses the same logic for describing designs and properties. This logic combines the for-all and exists-some quantifiers of first-order logic with the operators of set theory and relational calculus.

The idea of modeling software designs with sets and relations had been pioneered in the Z language.[32] Alloy incorporated much of the power of Z, while simplifying the logic to make it more tractable.

Alloy allows only first-order structures, thus ruling out, for example,

Alloy was inspired by the successes and limitations of model checkers.

sets of sets and relations over sets. This changes how designs are modeled, but not what can be modeled; after all, relational databases have flourished despite being first order.

Taking advantage of this restriction, Alloy's operators are defined in a very general way so that most expressions can be written with just a few operators. The key operator is relational join, which in conventional mathematics only applies to binary relations, but in Alloy works on relations of any arity. By using a dot to represent the join operator, Alloy lets you write dereferencing expressions as you would in an object-oriented programming language, but gives these expressions a simple mathematical interpretation. So, as in Java, given an employee *e*, a relation *dept* that maps employees to departments, and a relation *manager* that maps departments to their managers, *e.dept.manager* would give the manager of *e*'s department. But unlike in Java, the expression will also work if *e* is a set of employees, or if *dept* can map an employee to multiple departments, giving the expected result—the set of managers of the set of departments that the employees *e* belong to. The expression *dept.manager* is well defined too, meaning the relation that maps employees to their managers. You can also navigate backward—writing *manager.m* for the department(s) that *e* manages.

(A note for readers interested in language design: This flexibility is achieved by treating all values as relations—a set being a relation with one column, and a scalar being a set with one element—and defining a join operator that applies uniformly over a pair of relations, irrespective of their arity. In contrast, other languages tend to have multiple operators, implicit coercions, or overloading to accommodate variants that Alloy unifies.)

Alloy was influenced also by modeling languages such as UML. Like the class diagrams of UML, Alloy makes it easy to describe a universe of objects as a classification tree, with each relation defined over nodes in this tree. Alloy's dot operator was inspired in part by the navigational expressions of the Object Constraint Language[39] (OCL) of UML, but by defining the dot as relational join, Alloy dramatically simplifies the semantics of navigation.

Small scope analysis. Even plain first-order logic (without relational operators) is not decidable. This means that no algorithm can exist that could analyze a software design written completely in a language like Alloy. So something has to give. You could make the language decidable, but that would cripple its expressive power and make it unable to express even the most basic properties of structures (although exciting progress has been made recently in applying decidable fragments of first-order logic to certain problems[29]). You could give up on automation, and require help from the user, but this eliminates most of the benefit of an analysis tool—analysis is no longer a reward for constructing a design model, but a major extra investment beyond modeling.

The other option is to somehow limit the analysis. Prior to Alloy, two approaches were popular. *Abstraction* reduces the analysis to a finite number of cases by introducing abstract values that each corresponds to an entire set of real values. This often results in false positives that are difficult to interpret. In practice, picking the right abstraction calls for considerable ingenuity. *Simulation* picks a finite number of cases, usually by random sampling, but it covers such a small part of the state space that subtle flaws elude detection.

Alloy offered a new approach: running all small tests. The designer specifies a *scope* that bounds each of the types in the specification. A scope of five, for example, would include tests involving at most five elements of each type: five network nodes, five packets, five identifiers, and so on.

The rationale for this is the *small scope hypothesis*, which asserts that most bugs can be demonstrated with small counterexamples. If you test for all small counterexamples, you are likely to find any bug. Many Alloy case studies have confirmed the hypothesis by performing an analysis in a variety of scopes and showing, retrospectively, that a small scope would have sufficed to find all the bugs discovered.

Translation to SAT. Even with small scopes, the state space of an Alloy model is fiendishly large. The state comprises a collection of variables whose values are relations. Just one binary relation in a scope of five has $5 \times 5 = 25$ possible edges, and thus 2^{25} possible values. A very small design might have five such relations, giving $(2^{25})^5$ possible states—about 10^{37} states. Even checking a billion cases per second, such an analysis would take many times the age of the universe.

Alloy therefore does not perform an explicit search, but instead translates the design problem to a *satisfiability problem* whose variables are not relations but simple bits. By flipping bits individually, a satisfiability (SAT) solver can usually find a solution (if there is one) or show that none exists by examining only a tiny portion of the space.

Alloy's analysis tool is essentially a compiler to SAT, which allows it to exploit the latest advances in SAT solvers. The success of SAT solvers has been a remarkable story in computer science—theoreticians had shown that SAT was inherently intractable, but it turned out that most of the cases that appeared in practice could be solved efficiently. So SAT went from being the archetypal insoluble problem used to demonstrate the infeasibility of other problems to being a soluble problem that other problems could be translated to. Alloy also applies a variety of tactics to reduce the problem prior to solving, most notably adding *symmetry breaking constraints* that save the SAT solver from considering cases equivalent to one another.

Example: Modeling Origins

To see Alloy in action, let's explore the design of an origin-tracking mechanism for Web browsers. The model shown here is a toy version of a real model that exposed several serious flaws in browser security.[1] While it cuts corners and is unrealistic in some respects, it does capture the spirit and style of the original model, and is fairly representative of how Alloy is often used.

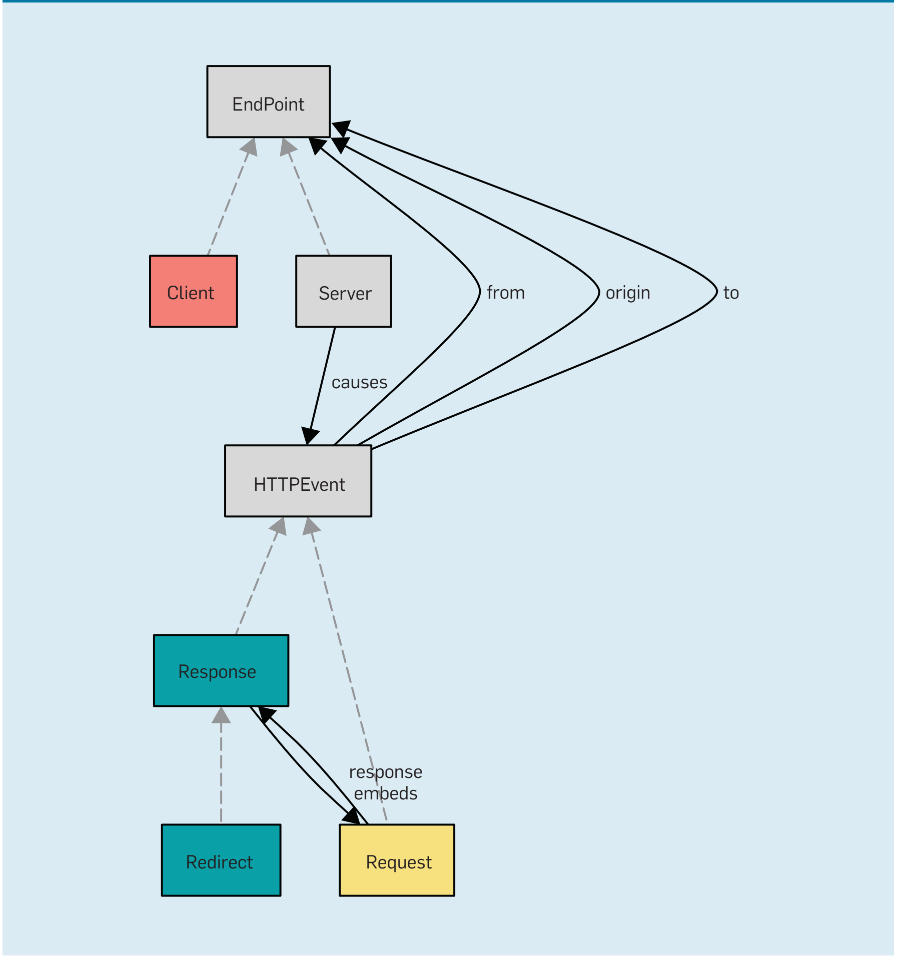
For those unfamiliar with browser security, cross-site request forgery (CSRF) is a pernicious and subtle attack in which a malicious script running in a page the user has loaded makes a hidden and unwanted request to a website for which the user is already authenticated. This may happen either because the user was enticed to load a page from a malicious server, or because a supposedly safe server was the subject of a cross-site scripting attack, and served a page containing a malicious script. Such a script can issue any request the user can issue; one of the first CSRF vulnerabilities to be discovered, for example, allowed an attacker to change the delivery address for the user's account in a DVD rental site. What makes CSRF particularly insidious is that the browser sends authentication credentials stored as cookies spontaneously when a request is issued, whether that request is made explicitly by the user or programmatically by a script.

One way to counter CSRF is to track the origins of all responses received from servers. In this example, the browser would mark the malicious script as originating at the malicious or compromised server. The subsequent request made by that script to the rental site server—the target of the attack—

Figure 1. Structure declarations.

```
1  abstract sig EndPoint { }

2  sig Server extends EndPoint {
3    causes: set HTTPEvent
4    }

5  sig Client extends EndPoint { }

6  abstract sig HTTPEvent {
7    from, to, origin: EndPoint
8    }

9  sig Request extends HTTPEvent {
10   response: lone Response
11   }

12  sig Response extends HTTPEvent {
13    embeds: set Request
14    }

15  sig Redirect extends Response {
16    }
```

Figure 2. Data model from declarations.



would be labeled as having this other origin. The target server can be configured so it only accepts requests that originate directly from the user (for example, by the user entering the URL for the request in the address bar), or from the target server itself (for example, from a script embedded in a page it previously sent). As always, the devil is in the details, and we shall see that a plausible design of this mechanism turns out to be flawed.

Some features to look out for in this model, which distinguish Alloy from many other approaches, include:

▸ A rich structure of objects, classification, and relationships;

▸ Constraints in a simple logic that exploits the relations and sets of the structure, avoiding the kind of low-level structures (arrays and indices especially) that are often required in model checkers;

▸ Capturing dynamic behavior without any need for a built-in notion of time or state;

▸ Intended properties to check expressed in the same language as the model itself; and,

▸ An abstract style of modeling that includes only those aspects essential to the problem at hand.

We start by declaring a collection of *signatures* (as illustrated in Figure 1). A signature introduces a set of objects and some fields that relate them to other objects. So *Server*, for example, will represent the set of server nodes, and has a field *causes* that associates each server with the set of HTTP events that it causes.

Keywords (or their omission) indicate the multiplicity of the relations between objects: thus each HTTP event has exactly one *from* endpoint, one to endpoint, and one *origin* endpoint (line 7); each request has at most one response (line 10, with *lone* being read as "less than or equal to one"); and each response embeds any number of requests (line 13).

Mathematically speaking, objects are just atomic identifiers without any internal structure. The *causes* relation includes tuples of the form (*s*, *e*) where the value of *s* is some atomic identifier representing a server object, and the value of *e* is some atomic identifier representing an event.

Fields are declared in signatures to allow a kind of object-oriented mindset. Alloy supports this by resolving field names contextually (so that field names need not be globally unique), and by allowing "signature facts" (not used here) that are implicitly scoped over the elements of a signature and their fields. But don't be misled into thinking there is some kind of fancy object semantics here. The signature structure is only a convenience, and just introduces a set and some relations.

The *extends* keyword defines one signature as a subset of another. An *abstract* signature has no elements that do not belong to a child signature, and the extensions of a signature are disjoint. So the declarations of *EndPoint*, *Server*, and *Client* imply the set of endpoints is partitioned into servers and clients: no server is also a client, and there is no endpoint that is neither client nor server. A relation defined over a set applies over its subsets too, so the declaration of *from*, for example, which says that every HTTP event is from a single endpoint, implies the same is true for every request and response. (Alloy is best viewed as untyped. It turns out that conventional programming language types are far too restrictive for a modeling language. Alloy thus allows expressions such as *HTTPEvent.response*, denoting the set of responses to any events, but its type checker rejects an expression such as *Request.embeds* that always denotes an empty set.[12])

The Alloy Analyzer can generate a graphical representation of the sets and relations from the signature declarations (see Figure 2); this is just an alternative view and involves no analysis.

Moving to the substance of what the model actually means:

▸ The *from* and *to* fields are just the source and destination of the event's packet.

▸ For a response *r*, the expression *r.embeds* denotes a set of requests that are embedded as JavaScript in the response; when that response is loaded into the browser, the requests are executed spontaneously.

▸ A *Redirect* is a special kind of response that indicates that a resource has moved, and spontaneously issues a request to a different server; this second request is modeled as an embedded request in the redirect response.

▸ The *origin* of an event is a notion computed by the browser as a means of preventing cross-site attacks. As we will see later, the idea is that a server may choose to reject an event unless it originated at that server or at a browser.

▸ The *cause* of an event is not part of the actual state of the mechanism. It is introduced in order to express the essential design property: that an evil server cannot cause a client to send a request to a good server.

Now let's look at the constraints (as shown in Figure 3). If there were no constraints, any behavior would be possible; adding constraints restricts the behavior to include only those that are intended by design.

The constraints are grouped into separate named *facts* to make the model more understandable:

▸ The *Directions* fact contains two constraints. The first says that every request is from, and every response is to, a client; the second says that every request is to, and every response is from, a server. These kinds of constraints can be written in many ways. Here I have chosen to use expressions denoting sets of endpoints—for example, *Request.from* for the set of endpoints that requests are from, but could just as well have written a constraint like:

```
from in
Request -> Client + Response -> Server
```

to say the from relation maps requests to clients and responses to servers. Or in a more familiar but less succinct style, we could have used quantifiers:

```
all r: Request | r.from in Client
all r: Response | r.from in Server
```

(constraining only the range of the relations, which is sufficient in this case since the declarations already constrain their domains).

▸ The *RequestResponse* fact defines some basic properties of how requests and responses work: that every response is associated with exactly one request (line 22); that every response is to the endpoint its request was from, and from the endpoint its request was to (line 23); and that a request cannot be embedded in a response to itself (line 24). Two expressions in these constraints merit explanation. The expression *response.r* exploits the flexibility of the join operator to navigate backward from the response *r* to the request it responds to; it could equivalently be written *r.~response* using the transpose operator ~. The expression *r.^(response.embeds)* starts with the request *r*, and then applies to it one or more navigations (using the closure operator ^) of following the *response* and *mbeds* relations, as if we had written instead the infinite expression

```
r.response.embeds
+ r.response.embeds
      .response.embeds
+ r.response.embeds
      .response.embeds
      .response.embeds
+ ...
```

defining the requests embedded in the response to *r*, the requests embedded in the response to the requests embedded in the response to *r*, and so on. (Equivalently, *r.^p* is the set of nodes reachable from *r* in the graph whose edges correspond to the relation *p*.)

▸ The *Causality* fact defines the *causes* relation. It says that an event is caused by a server if and only if it is from that server, or is embedded in a response that the server causes.

▸ The *Origin* fact describes the origin-tracking mechanism. Each constraint defines the origin of a different kind of HTTP event. The first (line 31) says that every embedded request e has the same origin as the response r that it is embedded in. The second (line 32) defines the origin of a response: it says if the response is a redirect, it has the same origin as the original request, and otherwise its origin is the server that the response came from. The third (line 33) handles a request that is not embedded: its origin is the endpoint it comes from (which will usually be the browser).

Finally, *EnforceOrigins* is a *predicate* that can be applied to a server, indicating that it chooses to enforce the origin header, allowing incoming requests only if they originate at that server, or at the client that sent the request.

With all this in place—the structure of endpoints and messages, the rules about how origins are computed and used, and the definition of causality—we can define a design property to check (as illustrated in Figure 4).

The keyword *check* introduces a command that can be executed. This command instructs the Alloy Analyzer to search for a refutation for the given constraint. In this case, the constraint asserts the nonexistence of a cross-site request forgery attack; refuting

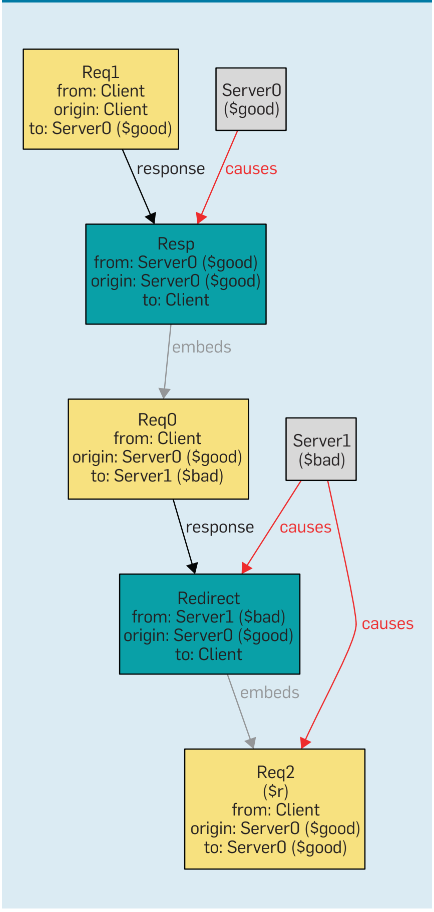
Figure 3. Fact and predicate declarations.

```
17  fact Directions {
18        Request.from + Response.to in Client
19        Request.to + Response.from in Server
20        }

21  fact RequestResponse {
22        all r: Response | one response.r
23        all r: Response | r.to = response.r.from and r.from = response.r.to
24        all r: Request | r not in r.^(response.embeds)
25        }

26  fact Causality {
27        all e: HTTPEvent, s: Server | e in s.causes iff
28              e.from = s or some r: Response | e in r.embeds and r in s.causes
29        }

30  fact Origin {
31        all r: Response, e: r.embeds |  e.origin = r.origin
32        all r: Response | r.origin = (r in Redirect implies response.r.origin else r.from)
33        all r: Request | no embeds.r implies r.origin in r.from
34        }

35  pred EnforceOrigins (s: Server) {
36        all r: Request | r.to = s implies r.origin = r.to or r.origin = r.from
37        }
```

Figure 4. Check command.

```
38  check {
39          no good, bad: Server {
40                  good.EnforceOrigins
41                  no r: Request | r.to = bad and r.origin in Client
42                  some r: Request | r.to = good and r in bad.causes
43          }
44  } for 5
```

Figure 5. Counterexample for check of Figure 4.



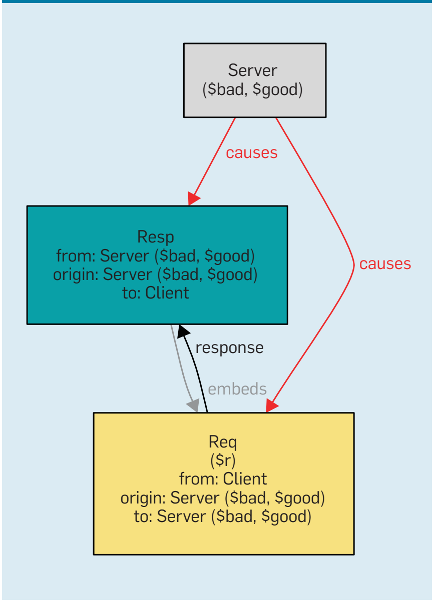
this will show that the origin mechanism is not designed correctly, and an attack is possible.

The constraint says that there are no two servers, *good* and *bad*, such that the good server enforces the origin header (line 40), there are no requests sent directly to the bad server that originate in the client (line 41), and yet there is some request to the good server that was caused by the bad server (line 42).

Analysis Results: Finding Bugs

The Alloy Analyzer finds a counterexample (see Figure 5) almost instantaneously—in 30ms on my 2012 MacBook (with a 2.6GHz i7 processor and 16GB of RAM).

Figure 6. A bogus counterexample.



The counterexample can be displayed in various ways—as text, a table, or a graph whose appearance can be customized. I've chosen the graph option, and have selected which objects are to appear as nodes (just the events and the servers), which relations are to appear as edges (those between events, and causes), and I've picked colors for the sets and relations. I have also chosen to use the Skolem constants (witnesses that the analyzer finds for the quantified variables) good and bad to label the servers.

Reading the graph from the top, looking just at the large rectangles representing the HTTP events, we see a request (*Req1*) was sent from a client to the good server. The response (*Resp*) embeds a request (*Req0*) that is sent to the bad server; this is a cross-site request, which will not be rejected because the bad server accepts incoming requests irrespective of origin. The bad server's response is to send a redirect whose embedded request (*Req2*) is received by the good server. (Note that the numbering of objects is arbitrary: *Req1* actually happens before *Req0*.)

Looking at the server nodes and the events they cause, we see that, as expected, the good server caused the response to the first request, and the bad server caused the redirect and its subsequent embedded request. The problem is the mismatch between cause and origin in the last request (*Req2*): we can see that it was *caused* by the bad server, but it was labeled as originating at the good server. In other words, the origin tracking design is allowing a cross-site request forgery by incorrectly identifying the origin of the request in the redirect.

The solution to this problem turns out to be non-trivial. Updating the origin header after each redirect would fail for websites that offer open redirection; a better solution is to list a chain of endpoints in the origin header.[1]

Agile Modeling

As mentioned earlier, our model is representative of many Alloy models. But the way I presented it was potentially misleading. In practice, users of Alloy don't construct a model in its entirety and then check its properties. Instead, they proceed in a more agile way, growing the model and simulating and checking it as they go.

Take, for example, the constraint on line 24 of Figure 3. Initially, I had not actually noticed the need for this constraint. But when I ran the check for the first time (without this constraint), the analyzer presented me with counterexamples such as the one shown in Figure 6, in which the response to a request is the very response in which the request is embedded!

One way to build a model exploiting Alloy's ability to express and analyze very partial models is to add one constraint at a time, exploring its effect. You do not need a property to check; you can just ask for an instance of the model satisfying all the constraints.

Doing this even before any explicit constraints have been included is very helpful. You can run just the data model by itself and see a series of instances that satisfy the constraints implicit in the declarations. Often doing this alone exposes some interesting issues. In this case, the first few instances include examples with no HTTP events, and with requests and responses that are disconnected.

To get more representative instances, you can specify an additional constraint to be satisfied. For example, the command

```
run (some response)
```

will show instances in which the *response* relation has some tuples. The first one generated (Figure 7) shows a request with a response that is a redirect from the same source as the request, and sent to an endpoint that is also its origin, and it includes an orphaned redirect unrelated to any request! These anomalies immediately suggest enrichments of the model.

When we developed Alloy, we underestimated the value of this kind of simulation. As we experimented with Alloy, however, we came to realize how helpful it is to have a tool that can generate provocative examples. These examples invariably expose basic misunderstandings, not only about what's being modeled but also about which properties matter. It's essential that Alloy provides this simulation for free: in particular, you do not need to formulate anything like a test case, which would defeat the whole point.

Growing a model in a declarative language like Alloy is very different from growing a program in a conventional programming language. A program starts with no behaviors at all, and as you add code, new behaviors become possible. With Alloy, it's the opposite. The empty model, since it lacks any constraints, allows every possible behavior; as you add constraints, behaviors are eliminated.

This allows a powerful style of incremental development in which you only add constraints that are absolutely essential for the task at hand—whether that is eliminating pathological cases or ensuring a design property holds.

Typically a model includes both a description of the mechanism being designed and some assumptions about the environment in which it operates. Our example model does not separate these rigorously, but where brevity is not such a pressing concern, it would be wise to do so. We could separate, for example, the constraints that model the setting and checking of the origin field from those that describe what kinds of requests and responses are possible.

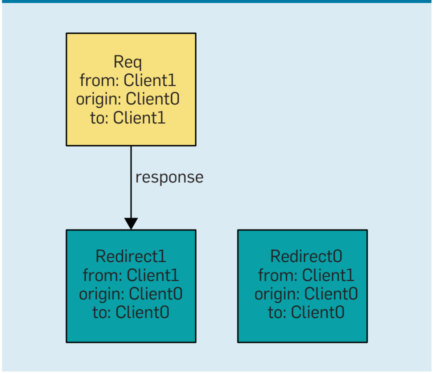
Like the class diagrams of UML, Alloy makes it easy to describe a universe of objects as a classification tree, with each relation defined over nodes in this tree.

Obviously, the less you assume about the environment, the better, since every assumption you make is a risk (as it may turn out to be untrue). In our model, for example, we do not require every request to have a response. It would be easy to do—just change the declaration of `response` in line 10 of Figure 1 by dropping the `lone` keyword—but would only make the result of the analysis less general. Likewise, the less you constrain the mechanism, the better. Allowing multiple behaviors gives implementation freedom, which is especially important in a distributed setting.

Simulation matters for a more profound reason. Verification—that is, checking properties—is often overrated in its ability to prevent failure. As Christopher Alexander explains,[2] designed artifacts usually fail to meet their purposes not because specifications are violated but because specifications are unknown. The "unknown unknowns" of a software design are invariably discovered when the design is finally deployed, but can often be exposed earlier by simulation, especially in the hands of an imaginative designer.

Verification, in contrast, is too narrowly focused to produce such discoveries. This is not to say property checking is not useful—it's especially valuable when a property can be assured with high confidence using a tool such as Alloy or a model checker or theorem prover (rather than by testing). But its value is always contingent on the sufficiency of the property itself, and

Figure 7. A simulated instance.



techniques that help you explore properties have an important role to play.

Uses of Alloy

Hundreds of papers have reported on applications of Alloy in a wide variety of settings. Here are some examples to give a better idea of how Alloy has been used:

Critical systems. A team at the University of Washington constructed a dependability case[18] for a neutron radiotherapy installation. The team devised an ingenious technique for verifying properties of code against specifications using lightweight, pluggable checkers. The end-to-end dependability case was assembled in Alloy from the code specifications, properties of the equipment and environment, and the expected properties, and then checked using the Alloy Analyzer. The analysis found several safety-critical flaws in the latest version of the control software, which the researchers were able to correct prior to its deployment. For a full description, see a recent research report[30] and additional information on the project's website.[36]

Network protocols. Pamela Zave, a researcher at AT&T, has been using Alloy for many years to construct and analyze models of networking as well as for designing a new unifying network architecture. In a major case study, she analyzed Chord, a distributed hash table for peer-to-peer applications. The original paper on Chord[33]—one of the most widely cited papers in computer science—notes that an innovation of Chord was its relative simplicity, and consequently the confidence users can have in its correctness. By modeling and analyzing the protocol in Alloy, Zave showed that the Chord protocol was not, in fact, correct, and she was able to develop a fixed version that maintains its simplicity and elegance while guaranteeing correct behavior.[43] Zave also used the explicit model checker SPIN[14] in this work, and wrote an insightful article explaining the relative merits of the two tools, and how she used them in tandem.[42]

Web security. The demonstration example of this article is drawn from a real study performed by a research group at UC Berkeley and Stanford.[1] The group constructed a library of Alloy models to capture various aspects of Web security mechanisms, and then analyzed five different mechanisms, including: WebAuth, a Web-based authentication protocol based on Kerberos deployed at several universities including Stanford; HTML5 forms; the Cross-Origin Resource Sharing protocol; and proposed designs for using the referrer header and the origin header to foil cross-site attacks (of which the last is the basis for the example here). The base library was written in 2,000 lines of Alloy; the various mechanisms required between 20 and 214 extra lines; and every bug was found within two minutes and a scope of 8. Two previously known vulnerabilities were confirmed by the analysis, and three new ones discovered.

As we experimented with Alloy, we came to realize how helpful it is to have a tool that can generate provocative examples.

Memory models. John Wickerson and his colleagues have shown that four common tasks in the design of memory models—generating conformance tests, comparing two memory models, checking compiler optimizations, and checking compiler mappings—can all be framed as constraint satisfaction problems in Alloy.[41] They were able to reproduce automatically several results for C11 (the memory model introduced in 2011 for C and C++) and common compiler optimizations associated with it, for the memory models of the IBM Power and Intel x86 chips, and for compiler mappings from OpenCL to AMD-style GPUs. They then used their technique to develop and check a new memory model for Nvidia GPUs.

Code verification. Alloy can also be used to verify code by translating the body of a function into Alloy, and asking it to find a behavior of the function that violates its specification. Greg Dennis built a tool called Forge that wraps Alloy so it can be applied directly to Java code annotated with JML specifications. In a case study application,[10] he checked a variety of implementations of the Java collections list interface, and found bugs in one (a GNU Trove implementation). Dennis also applied his tool to KOA, an electronic voting system used in the Netherlands that was annotated with JML specifications and had previously been analyzed with a theorem-proving tool, and found several functions that did not satisfy their specifications.[11]

Civil engineering. In one of the more innovative applications of Alloy, John Baugh and his colleagues have been applying Alloy to problems in large-scale physical simulation. They designed an extension to ADCIRC—an ocean circulation model widely used by the U.S. Army Corps of Engineers and others for simulating hurricane storm surge—that introduces a notion of subdomains to allow more localized computation of changes (and thus reduced overall computational effort). Their extension, which has been incorporated into the official ADCIRC release, was modeled and verified in Alloy.[7]

Alloy as a backend. Because Alloy offers a small and expressive logic, along with a powerful analyzer, it has been exploited as a backend in many different tools. Developers have often used Alloy's own engine, Kodkod,[34] directly, rather than the API of Alloy itself, because it offers a simpler programmatic interface with the ability to set bounds on relations, improving performance. Jasmin Blanchette's Nitpick tool,[8] for example, uses Kodkod to find counterexamples in Isabelle/HOL, saving the user the trouble of trying to prove a theorem that is not true, and the Margrave tool[26] analyzes firewall configurations. Last year, a team from Princeton and Nvidia built a tool that uses Alloy to synthesize security attacks that exploit the Spectre and Meltdown vulnerabilities.[35]

Teaching. Alloy has been widely taught in undergraduate and graduate courses for many years. At the University of Minho in Portugal, Alcino Cunha teaches an annual course on formal methods using Alloy, and has developed a Web interface to present students with Alloy exercises (which are then automatically checked). At Brown University, Tim Nelson teaches Logic for Systems, which uses Alloy for modeling and analysis of system designs, and has become one of the most popular undergraduate classes. Because the Alloy language is very close to a pure relational logic, it has also been popular in the teaching of discrete mathematics, for example, in a course that Charles Wallace teaches at Michigan Technological University[38] and appearing as a chapter in a popular textbook.[15]

Alloy Extensions

Many extensions to Alloy—both to the language and to the tool—have been created. These offer a variety of improvements in expressiveness, performance, and usability. For the most part, these extensions have been mutually incompatible, but a new open source effort is now working to consolidate them. There are too many efforts to include here, so I focus on representatives of the main classes.

Higher-order solving. The Alloy Analyzer's constraint-solving mechanism cannot handle formulas with universal quantifications over relations—that is, problems that reduce to "find some relation P such that for every relation Q ..." This is exactly the form that many synthesis problems take, in which the relation P represents a structure to be synthesized, such as the abstract syntax tree of a program, and the relation Q represents the state space over which certain behaviors are to be verified. Alloy*[24] is an extension of Alloy that can solve such formulas, by generalizing a tactic known as counterexample-guided inductive synthesis that has been widely used in synthesis engines.

Temporal logic. Alloy has no built-in notion of time or dynamic behavior. On the one hand, this is an asset, because it keeps the language simple, and allows it to be used very flexibly. I exploited this in the example model here, where the flow of time is captured in the *response* relation that maps each request to its response. By adding a signature for state, Alloy supports the specification style common in languages such as B, VDM, and Z; and by adding a signature for events, Alloy allows analysis over traces that can be visualized as a series of snapshots. On the other hand, it would often be preferable to have dynamic features built into the language. Electrum[20] extends Alloy with a keyword *var* to indicate that a signature or field has a time-varying value, and with the quantifiers of linear temporal logic (which fit elegantly with Alloy's traditional quantifiers). DynAlloy[31] offers similar functionality, but using dynamic logic instead, and is the basis of an impressive code analysis tool called TACO[13] that outperforms Forge (mentioned earlier) by employing domain-specific optimizations. No extension of Alloy, however, has yet addressed the problem of combining Alloy's capacity for structural analysis with the ability of traditional model checkers to explore long traces, so Alloy analyses are still typically limited to short traces.

Instance generation. The result of an Alloy analysis is not one but an entire set of solutions to a constraint-solving problem, each of which represents either a positive example of a scenario, or a negative example, showing how the design fails to meet some property. The order in which these appear is somewhat arbitrary, being determined both by how the problem is encoded and the tactics of the backend SAT solver. Since SAT solvers tend to try false before true values, the instances generated tend to be small ones—with few nodes and edges. This is often desirable, but is not always ideal. Various extensions to the Alloy Analyzer provide more control over the order in which instances appear. Aluminum[28] presents only minimal scenarios in which every relation tuple is needed to satisfy the constraints, and lets the user add new tuples, automatically compensating with a (minimal) set of additional tuples required for consistency. Amalgam[27] lets users ask about the provenance of an instance, indicating which sub formula is responsible for requiring (or forbidding) a particular tuple in the instance. Another extension[21] of the Alloy Analyzer generates minimal and maximal instances, and choosing a next instance that is as close to, or as far away from, the current instance as possible.

Better numerics. Alloy handles numerical operations by treating numbers as bit strings. This has the advantage of fitting into the SAT solving paradigm smoothly, and it allows a good repertoire of integer operations. But the analysis scales poorly, making Alloy unsuitable for heavily numeric applications. The finite scopes of Alloy can also be an issue when a designer would like numbers to be unbounded. A possible solution is to replace the SAT backend with an SMT backend instead. This is challenging because SMT solvers have not traditionally supported relational operators. Nevertheless, a team at the University of

Iowa has recently extended CVC4, a leading SMT solver, with a theory of finite relations, and has promisingly demonstrated its application to some Alloy problems.[23]

Configurations. Many Alloy models contain two loosely coupled parts, one defining a configuration (say of a network) and the other the behavior (say of sending packets). By iterating through configurations and analyzing each independently, one can often dramatically reduce analysis time.[22] In some applications, a configuration is already fully or partially known, and the goal is to complete the instance—in which case searching for the configuration is a wasted effort. Kodkod, Alloy's engine, allows the explicit definition of a "partial instance" to support this, but in Alloy itself, this notion is not well supported (and relies on a heuristic for extracting partial instances from formulas in a certain form). Researchers have therefore proposed a language extension[25] to allow partial instances to be defined directly in Alloy itself.

How to Try Alloy

The Alloy Analyzer[3] is a free download available for Mac, Windows, and Linux. The Alloy book[16] provides a gentle introduction to relational logic and to the Alloy language, gives many examples of Alloy models, and includes a reference manual and a comparison to other languages (both of which are available on the book's website[17]). The Alloy community answers questions tagged with the keyword *alloy* on StackOverflow, and hosts a discussion forum.[5] A variety of tutorials for learning Alloy are available online too, as well as blog posts with illustrative case studies and examples (for example, Kriens[19] and Wayne[40]). The model used in this article is available (along with its visualization theme) in the Alloy community's model repository.[4]

Acknowledgments

I am very grateful to David Chemouil, Alcino Cunha, Peter Kriens, Shriram Krishnamurthi, Jay Parlar, Emina Torlak, Hillel Wayne, Pamela Zave, and the anonymous reviewers, whose suggestions improved this article greatly; to Moshe Vardi, who encouraged me to write it; and to Devdatta Akhawe, Adam Barth, Peifung E. Lam, John Mitchell, and Dawn Song, whose work formed the basis of the example used in the article. Thank you also to the many members of the Alloy community who have contributed to Alloy over the years. C

References

1. Akhawe, D., Barth, A., Lam, P.E., Mitchell, J. and Song, D. Towards a formal foundation of Web security. In *Proceedings of the 23rd IEEE Computer Security Foundations Symp.* Edinburgh, 2010, 290–304.
2. Alexander, C. *Notes on the Synthesis of Form.* Harvard University Press, 1964.
3. Alloy Tools; http://alloytools.org.
4. Alloy Models repository; https://github.com/AlloyTools/models
5. Alloy discussion forum; https://groups.google.com/forum/#!forum/alloytools
6. Barth, A., Jackson, C., and Mitchell, J.C. Robust defenses for cross-site request forgery. In *Proceedings of the 15th ACM Conf. on Computer and Communications Security*. ACM, 2008, 75–88.
7. Baugh, J. and Altuntas, A. Formal methods and finite element analysis of hurricane storm surge: A case study in software verification. *Science of Computer Programming 158* (2018), 100–121.
8. Blanchette, J. and Nipkow, T. Nitpick: A counterexample generator for higher-order logic based on a relational model finder. In *Proceedings of the 1st Intern.Conf. Interactive Theorem Proving*. M. Kaufmann and L.C. Paulson, eds. *LNCS 6172* (2010). Springer, 131–146.
9. Burch, J.R., Clarke, E.M., McMillan, K.L., Dill, D.L. and Hwang, L.J. Symbolic model checking: 10 states and beyond. In *Proceedings of the 5th Annual Symp. Logic in Computer Science*. (Philadelphia, PA, USA, June 4–7, 1990), 428–439.
10. Dennis, G., Chang, F. and Jackson, D. Modular verification of code with SAT. In *Proceedings of the Intern. Symp. Software Testing and Analysis*. (Portland, ME, July 2006).
11. Dennis, G., Yessenov, K. and Jackson, D. Bounded verification of voting software. In *Proceedings of the 2nd IFIP Working Conf. Verified Software: Theories, Tools, and Experiments.* (Toronto, Canada, Oct. 2008).
12. Edwards, J., Jackson, D. and Torlak, E. A type system for object models. In *Proceedings of the 12th ACM SIGSOFT Intern. Symp. Foundations of Software Engineering* (Newport Beach, CA, USA, Oct. 31— Nov. 6, 2004), 189–199.
13. Galeotti, J.P., Rosner, N., Lopez Pombo, C.G. and Frias, M.F. TACO: Efficient SAT-based bounded verification using symmetry breaking and tight bounds. *IEEE Trans. Softw. Eng. 39*, 9 (Sept.2013), 1283–1307.
14. Holzmann, G.J. *The Spin Model Checker: Primer and Reference Manual*, Addison Wesley, 2003.
15. Huth, M. and Ryan, M. *Logic in Computer Science: Modeling and Reasoning about Systems.* Cambridge University Press, 2004.
16. Jackson, D. *Software Abstractions.* MIT Press, Second edition, 2012.
17. Jackson, D. *Software Abstractions*; http://softwareabstractions.org.
18. Jackson, D., Thomas, M. and Millett, L.I. eds. *Software For Dependable Systems: Sufficient Evidence?* Committee on Certifiably Dependable Software Systems, Computer Science and Telecommunications Board, Division on Engineering and Physical Sciences. National Research Council of the National Academies. The National Academies Press, Washington, D.C., 2007.
19. Kriens, P. *JPMS, The Sequel*; http://aqute.biz/2017/06/14/jpms-the-sequel.html
20. Macedo, N., Brunel, J., Chemouil, D., Cunha, A. and Kuperberg, D. Lightweight specification and analysis of dynamic systems with rich configurations. In *Proceedings of the 24th ACM SIGSOFT Intern. Symp. Foundations of Software Engineering* (Seattle, WA, USA, 2016), 373–383.
21. Macedo, N., Cunha, A. and Guimaraes, T. Exploring scenario exploration. *Fundamental Approaches to Software Engineering.* A. Egyed and I. Schaefer, eds. *Lecture Notes in Computer Science 9033.* Springer, Berlin, Heidelberg.
22. Macedo, N., Cunha, A. and Pessoa, E. Exploiting partial knowledge for efficient model analysis. In *Proceedings of the 15th Intern. Symp. Automated Technology for Verification and Analysis.* Springer, 2017, 344–362.
23. Meng, B., Reynolds, A., Tinelli, C., and Barrett, C. Relational Constraint Solving in SMT. In *Proceedings of the 26th Intern. Conf. Automated Deduction.* (Gothenburg, Sweden, 2017) L. de Moura, ed. Springer.
24. Milicevic, A., Near, J.P., Kang E. and Jackson, D. Alloy*: A general-purpose higher-order relational constraint solver. *Formal Methods in System Design*, 2017, 1–32.
25. Montaghami, V. and Rayside, D. Extending alloy with partial instances. In *Proceedings of the 3rd Intern.* Conf. Abstract State Machines, Alloy, B, VDM, and Z. 2012, 122–135.
26. Nelson, T., Barratt, C., Dougherty, D.J., Fisler, K. and Krishnamurthi, S. The Margrave tool for firewall analysis. In *Proceedings of the 24th USENIX Large Installation System Administration Conference* (San Jose, CA, 2010).
27. Nelson, T., Danas, N., Dougherty, D.J., and Krishnamurthi, S. The Power of Why and Why Not: Enriching Scenario Exploration with Provenance. *Joint European Software Engineering Conference and ACM SIGSOFT Symposium on the Foundations of Software Engineering*, 2017.
28. Nelson, T., Saghafi, S., Dougherty, D.J., Fisler, K., and Krishnamurthi, S. Aluminum: Principled scenario exploration through minimality. In *Proceedings of the Intern. Conf. Software Engineering*, 2013.
29. Padon, O., Losa, G., Sagiv, M. and Shoham S. Paxos made EPR: Decidable reasoning about distributed protocols. In *Proceedings of the OOPSLA 2017* (Vancouver, 2017).
30. Pernsteiner, S. et al. Investigating safety of a radiotherapy machine using system models with pluggable checkers. *Computer Aided Verification LNCS* 9780. Springer.
31. Regis, G. et al. DynAlloy Analyzer: A tool for the specification and analysis of Alloy models with dynamic behaviour. In *Proceedings of the 11th Joint Meeting on Foundations of Software Engineering.* ACM, New York, NY, 2017, 969–973.
32. Spivey, J.M. *The Z Notation: A Reference Manual* (2nd ed.), Prentice Hall, 1992.
33. Stoica, I. et al. Chord: A scalable peer-to-peer lookup protocol for Internet applications. *IEEE/ACM Trans. Networking 11*, 1 (2003), 17–32.
34. Torlak, E. and Jackson, D. Kodkod: A relational model finder. In *Proceedings of the 13th Intern. Conf. Tools and Algorithms for the Construction and Analysis of Systems* (Braga, Portugal, 2007), 632–647.
35. Trippel, C., Lustig, D. and Martonosi, M. *MeltdownPrime and SpectrePrime: Automatically Synthesized Attacks Exploiting Invalidation-Based Coherence Protocols.* Feb. 2018; arX- iv:1802.03802.
36. University of Washington. *PLSE Neutrons*; http:neutrons.uwplse.org/
37. Visser, W., Havelund, K., Brat, G., Park, S.J. and Lerda, F. Model checking programs. *Automated Software Engineering J. 10*, 2 (Apr. 2003).
38. Wallace, C. Learning Discrete Structures Interactively with Alloy. In *Proceedings of the 49th ACM Tech. Symp. Computer Science Education* (Baltimore, MD, Feb. 21–24, 2018), 1051–1051.
39. Warmer, J.B. and Kleppe, A.G. *The Object Constraint Language: Precise Modeling with UML.* Addison-Wesley, 1999.
40. Wayne, H. Personal blog; https://www.hillel-wayne.com
41. Wickerson, J, Batty, M., Sorensen, T. and Constantinides, G.A. Automatically comparing memory consistency models. In *Proceedings of the 44th ACM SIGPLAN Symp. Principles of Programming Languages* (Paris, France, 2017), 190–204.
42. Zave, P. A practical comparison of Alloy and Spin. *Formal Aspects of Computing 27* (2015), 239–253.
43. Zave, P. Reasoning about identifier spaces: How to make Chord correct. *IEEE Trans. Software Engineering 43*, 12 (Dec. 2017), 1144–1156.

Daniel Jackson is a professor of computer science and Associate Director of the Computer Science and Artificial Intelligence Laboratory at the Massachusetts Institute of Technology, Cambridge, MA, USA.

DOI:10.1145/3333613

EarSketch leverages the appeal of music to create a learning environment that allows students to be expressive with code.

BY JASON FREEMAN, BRIAN MAGERKO, DOUG EDWARDS, TOM MCKLIN, TANEISHA LEE, AND ROXANNE MOORE

EarSketch: Engaging Broad Populations in Computing Through Music

"LIKE, I THOUGHT coding was going to be boring and kind of just make me super-mad. It was going to be like tragic. But now that I've taken this class and I've seen all the things I can do with EarSketch and how that can be applied—like the same general concepts can be applied and expanded on to all these other aspects and different fields—it kind of opened up and made me kind of rethink my career choices like, 'Oh, maybe I actually want to pursue something in like IT or computer science.' Normally, you have like a one-sided opinion or view of coding. You don't really see it as being something creative and so personable. ... It just kind of opened up your world, like broadened your horizons in seeing all the career fields that actually use coding and how that plays a role in it, versus like this stereotypical view of what coding is."

This is a reflection from a high school student in an introductory computer science course. During a focus group, he discussed his changing perceptions of and interest in coding. This student's shift in perceptions about computing and its level of engagement, potential for creativity, and career relevance exemplifies the critical importance of students' early academic experiences with computing. In other words, an engaging and expressive introductory computing course can significantly impact students' intention to persist in the field.

EarSketch (Figure 1), the learning environment and curriculum this student used in his course, engages students by emphasizing the personally expressive role of computing in the domain of music. EarSketch students learn elements of computing and sample-based music composition (that is, composition using musical beats, samples, and effects). They write Python or JavaScript code to algorithmically create music in popular genres and use fundamental computing concepts such as loops, lists, and user-defined functions to manipulate musical samples, beats, and effects.

The computational thinking skills that underlie these activities have become central to how we create, communicate, experiment, evaluate, iter-

» **key insights**

- **Computer science learning environments and curricula focused on creative domains—such as music—can help to engage students in introductory computing courses and to broaden participation in computing.**
- **By providing students the opportunity to easily create personally expressive work through code, these learning environments can leverage the unique affordances of creative computing regardless of students' prior experiences.**
- **When students perceive a creative computing environment to be authentic and when they want to share what they have made with others, they may exhibit growth in their attitudes toward computing and their intention to persist in the field.**

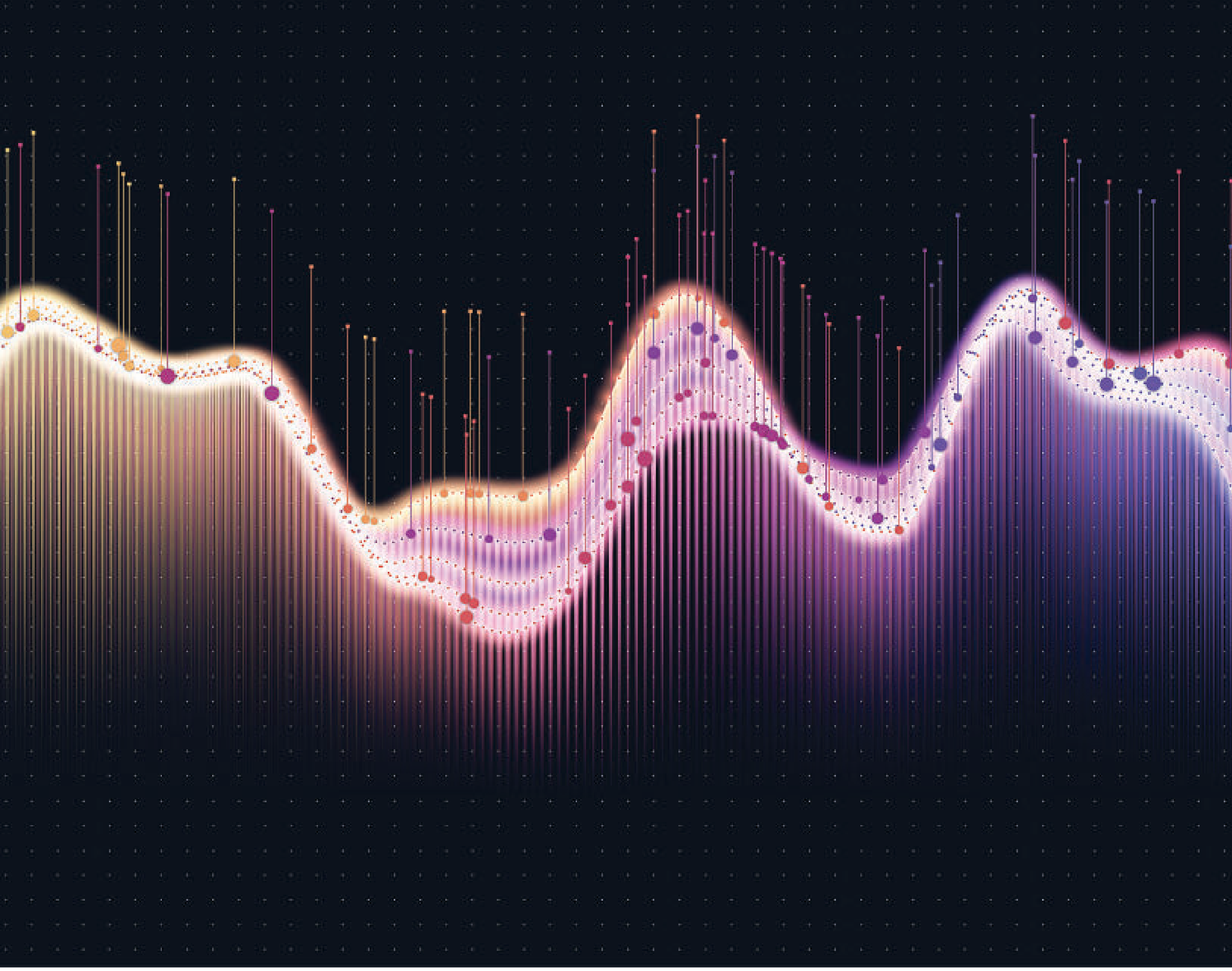


IMAGE BY GARRY KILLIAN

ate, and innovate in the 21st century.[39] Computer science is a core skill not only in a growing high-tech sector, but also for careers across many other domains; yet, computing is often seen by students as uncool[20] and approaches to teaching it may be uninspiring.[26] African American and Latino students, as well as women, are vastly underrepresented in computing courses as compared to their male Caucasian and Asian counterparts. (Demographic data from the Advanced Placement Computer Science A exam[10] clearly documents this trend.)

The integration of music into introductory computing education presents unique opportunities to engage students in the study of computing and to broaden participation in the field. Music is a ubiquitous part of human culture, with directly observable neurological foundations in the human brain.[31] Students dedicate an enormous portion of their daily lives to music listening and sharing, and these activities play a crucial role in forming their cultural and social identity.[2] A recent survey of high school students studying EarSketch, for example, reinforced the prevalence of music in students' lives: 59.8% of students reported spending three or more hours per day listening to music. Additionally, the rise of consumer-facing music software and apps, ranging from GarageBand to Magic Piano,[16] has made computer-based music creation a ubiquitous practice—even for users without prior training in music or music technology.

In addition to music's potential use as a hook to engage broad student populations in computing, pedagogical connections between the two disciplines abound. Many musical concepts, structures, and processes map easily and naturally to computational thinking. For example, the abstraction of code segments into functions parallels the repetition—with variation—of phrases and sections of music, and the sequential representation of characters in a string mimics the encoding of rhythmic data in a drum sequencer.

Our work with EarSketch leverages this tremendous potential of combining music and coding and embraces two overarching design priorities:

▸ EarSketch is designed to provide an "immediate opportunity to act"[7] and to be musically expressive, even for students (and teachers) who have no previous background in either music or computing. Anyone can quickly begin making compelling music in EarSketch with just a few lines of code and audio loops from an included library.

▸ EarSketch is designed so students will perceive it to be authentic[15,23,37] in both the computing and music domains. Its interface design and underlying functionality borrow heavily from standard music production and software development tools and practices. EarSketch uses programming languages that are pervasive in real-world computing practices. It also provides students with audio samples from popular genres created by well-known musicians that serve as the musical building blocks for their compositions.

In this article, we first summarize related work in music and computing. We then explain how we operationalize both the immediate opportunity to act and the perception of authenticity in a dual-domain—music plus computing—approach and we describe EarSketch's learning environment and curriculum within this framing. Finally, we summarize recent research findings with respect to these core ideas and to the impact of EarSketch on student engagement and intention to persist in computing.

Related Work

In algorithmic composition,[12] musicians define a process that generates the musical events (such as, notes or sound objects) of a composition. This practice contrasts with writing a score, in which musicians directly and linearly specify the properties of each individual musical event.

The practice of algorithmic composition includes precomputing examples (for example, Mozart's Musical Dice Game[18]), early experiments with composition on computers (for example, Lejaren Hiller and Leonard Isaacson's ILLIAC Suite[21]), domain-specific languages for computer music currently in widespread use (Max[32] and Supercollider[28]), and algorithmic features embedded in commercial music production software (Drummer in Apple's Logic[a]). Innovative signal processing and machine learning algorithms also underpin core techniques in audio production such as auto-tuning, time-stretching, and source separation.[41] Recent advances in machine learning have also inspired efforts to automate entire phases of the music creation process such as music generation (for example, Google's Magenta[22]) and audio mastering (Landr[b]), following from earlier work in both the analog (Electronium[8]) and digital (Experiments in Musical Intelligence[9]) realms.

Most learning environments for computer science support some music or audio functionality, dating back as far as Logo.[14] Popular learning environments for computing like Scratch[33] and Pencilcode[6] can play back audio files as well as lists of musical pitches and rhythms. These tools have been used to create sophisticated musical systems and performances.[35]

Other educational programming environments have been designed specifically for music. For example, JythonMusic is a Python programming environment and curriculum that supports the creation of both musical scores and interactive systems.[27] Sonic Pi is a Ruby-based programming environment and curriculum that focuses on live coding (that is, modifying code while it is executing to dynamically change the musical output).[1]

EarSketch draws inspiration from these and other creative coding projects that have demonstrated success in introductory computing education, but differs from prior work with respect to immediate opportunity to act and perceived authenticity.

Musical composition can be approached in terms of dimensions of musical objects (for example, pitch, harmony, timbre) as well as hierarchies of musical time (for example, microsound, sound object, and phrase).[34] Most algorithmic composition environments, whether designed for educational or professional use, focus on either the sound object level (such as lists of notes in Bau et al.[6] or Cope[9]), on the subsymbolic level (such as sound synthesis descriptors in McCartney[28]), or sometimes on both as in Puckette[32] and Aaron.[1] In contrast, EarSketch focuses primarily on the phrase level of music: students recombine audio loops—each several seconds in duration—to create a new composition. Remixing is the dominant compositional activity: programmatically arranging these loops in simultaneity and succession on a multi-track timeline and adding effects and automations to those tracks. In other words, EarSketch operates at a much higher level of abstraction than most other environments: it focuses more on working with longer sections of audio than on manipulating the individual musical elements that comprise each of those blocks.

By focusing on composition at this higher level of abstraction, EarSketch provides students an immediate opportunity to act in the musical domain. No prior experience reading music notation or playing an instrument is necessary, and EarSketch neither requires nor suggests that such skills are needed to make compelling music. While the curriculum does introduce key aspects of music technology (like multi-track editing and effects) and the basics of musical form and time, it avoids topics such as musical keys, scales, harmony, and notation. Because EarSketch is primarily taught within computer science classrooms, most teachers are not musicians and so there is no expectation that teachers have any background in music either. This high-level of abstraction in musical composition does sometimes constrain the modalities of creativity in EarSketch; compared to Manaris[27] or Pyknon,[c] EarSketch's design encourages the creation of repetitive music while discouraging the creation of lower-level musical content from scratch (such as melodies).

EarSketch not only offers an immediate opportunity to act in terms of these low barriers to entry, but also in terms of the speed with which students can create music. Within an hour of learning EarSketch, students are already writing scripts that create complete musical songs and are iteratively extending and revising them based on their musical preferences and intuitions.

EarSketch is also designed to be perceived by students as authentic in both the computing and music domains. The authenticity of a learning experience, according to Lee and Butler,[23] is based on the interrelated authentic learning practices of: having personally meaningful learning experiences; learning that relates to the world outside of the learning context; learning that encourages thinking within a particular discipline (for example, music composition); and allowing for assessment that reflects the learning process. Thick authenticity, according to Shaffer and Resnick,[37] meets all of these

a https://www.apple.com/logic-pro/
b https://www.landr.com/
c http://kroger.github.io/pyknon/

requirements in a single approach/system. Guzdial and Tew[15] argue that it is more important students perceive a learning experience to be authentic than that they learn in a manner that is completely consistent with real-world practices.

Students perceive EarSketch to be authentic across computing and music domains (related research findings are discussed later). Learning with EarSketch is personally meaningful to students who can create music in styles and genres that they like. EarSketch's use of popular programming languages, its reliance on multi-track audio editing paradigms in its interface and API design, and its library of sounds created by well-known musicians (as we will explore next) emphasize the relationships between the learning environment and real-world practices in both the computing and music industries. The EarSketch curriculum builds upon these connections by incorporating appropriate computing and music skills and by assessing students through projects that further emphasize the real-world relevance of students' learning.

Unlike systems such as McCartney,[28] Puckette,[32] and Aaron,[1] EarSketch is not intended for use by algorithmic music practitioners and researchers. EarSketch's focus on immediate opportunity to act, a high level of abstraction, and a connection to multitrack audio editing paradigms leads to a feature set that fully supports an introductory computer science curriculum. The design resulting from these priorities, however, precludes support for lower-level audio synthesis, signal processing, and symbolic music manipulation features that are common across programming environments designed specifically for musicians creating algorithmic music. Ariza[3] provides a thorough overview of algorithmic computing environments designed for that distinct use context.

Here, we further describe the EarSketch learning environment and curriculum in the contexts of immediate opportunity to act and of perceived authenticity.

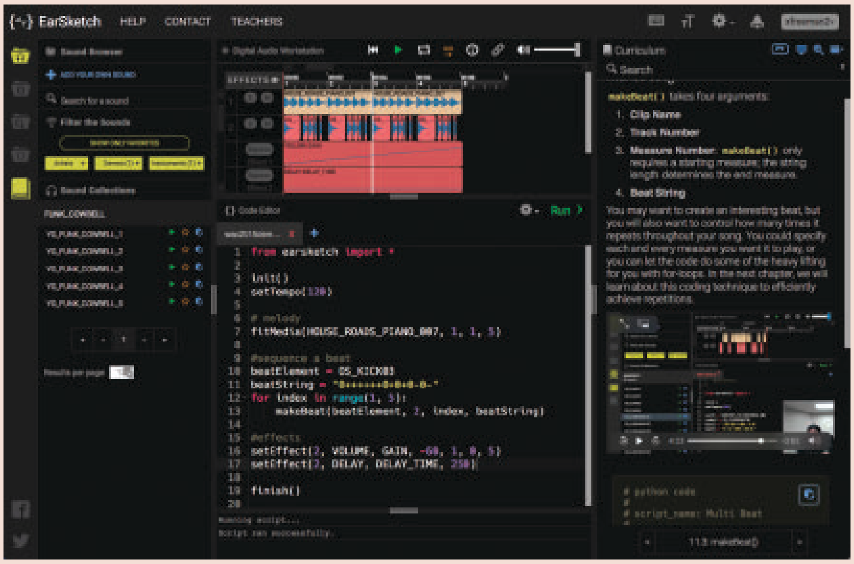
Learning Environment

The EarSketch learning environment is a browser-based application that uses modern Web standards and the Web Audio API[d] to integrate a code editor, language interpreter, digital audio workstation (DAW), loop library, and curriculum viewer within a single-window interface.[25] The interface (Figure 1) borrows common design cues from both IDEs and DAWs, such as a central code editor and audio view flanked by swappable sidebars for file management, sharing, and reference.

In the EarSketch code editor, students write code in Python or JavaScript, using either a text editor or a blocks-based visual code editor.[5] Regardless of language or editor chosen, they use the same application-programming interface (API) to create music. The use of programming lan-

d https://www.w3.org/TR/webaudio/

Figure 1. The EarSketch learning environment includes a sound browser (left), code editor (center bottom), digital audio workstation (center top), and curriculum browser (right).



```
from earsketch import *

init()
setTempo(120)

# melody
fitMedia(HOUSE_ROADS_PIANO_007, 1, 1, 5)

#sequence a beat
beatElement = OS_KICK03
beatString = "0+++++0+0+0-0-"
for index in range(1, 5):
    makeBeat(beatElement, 2, index, beatString)

#effects
setEffect(2, VOLUME, GAIN, -60, 1, 0, 5)
setEffect(2, DELAY, DELAY_TIME, 250)

finish()
```

```
Running script...
Script ran successfully.
```

makeBeat() takes four arguments:

1. Clip Name
2. Track Number
3. Measure Number: makeBeat() only requires a starting measure; the string length determines the end measure.
4. Beat String

You may want to create an interesting beat, but you will also want to control how many times it repeats throughout your song. You could specify each and every measure you want it to play, or you can let the code do some of the heavy lifting for you with for-loops. In the next chapter, we will learn about this coding technique to efficiently achieve repetitions.

```
# python code
#
# script_name: Multi Beat
```

Figure 2. Audio engineer Young Guru, who created many of the sounds in the EarSketch loop library, reviews an EarSketch student's project.

guages popular in real-world practice emphasizes the real-world dimension of authenticity, as well as the transferability of skills to other computational domains and to other educational and career contexts.

The code editor in Figure 1 shows a simple EarSketch program in Python that incorporates a few of the most common API functions. The `fitMedia()` function on line 7 places an audio loop on a particular track at specified start and end times, repeating the audio as necessary to fill the requested duration. The `setEffect()` functions on lines 16 and 17 add effects to a track. Line 17 adds a delay (that is, recurring echo) effect, and line 16 adds a volume fade-in in the opening measures of the song.

The `makeBeat()` function on line 13 is one of the few API functions that works at a sound object (note) level instead of the audio loop (phrase) level. Despite this, it still provides students an immediate opportunity to act because of its focus on rhythm rather than pitch. Following the paradigm of a step or drum sequencer, it divides a measure of music into steps, with the musical contents of each step represented by a character in a string. In this manner, a student can easily create a rhythm with a string, listen to it, and iteratively modify it until they are satisfied with the musical result. All of these EarSketch functions emphasize the disciplinary dimension of authenticity, encouraging students to think in terms of the multi-track paradigm that is ubiquitous in music creation and production.

The EarSketch API includes additional functions for tasks such as analyzing audio for its amplitude or brightness, importing files and images to use as datasets, and manipulating the strings used in `makeBeat()`. Because EarSketch is targeted at introductory computing students, the API does not support advanced computational features (for example, deep learning), and support for signal processing is limited to using 16 predefined audio effects.

When users run their code, the results of execution are displayed in a digital audio workstation (DAW) panel that closely mimics the multitrack displays found in music production software (see the center top of Figure 1). Students can see and hear the results of code execution, control playback with transport controls, export their audio for use in other music software, or share it directly online. Unlike conventional DAWs, they cannot directly modify audio or effects in the graphical interface: they must accomplish this through changes in code.

EarSketch focuses primarily on the phrase level of music: students recombine audio loops—each several seconds in duration—to create a new composition.

EarSketch includes ~4000 prerecorded sound samples accessible via a sound browser sidebar. The sound browser pane mimics the functionality of similar interface panels in DAWs, allowing users to search and filter sounds by artist, genre, and instrument. Sounds are grouped into collections that contain loops in the same style and key and are designed to fit well together. By using loops within the same collection, novice users are easily able to create music that is stylistically, harmonically, and rhythmically coherent, even without knowledge of the music theory behind these elements. The collections, which cover a wide range of popular genres (for example, hip hop, dubstep, EDM, and pop), were created by sound designer and electronic musician Richard Devine and Young Guru (Figure 2), Jay-Z's Grammy-nominated audio engineer and DJ. Students can also upload their own sounds, record sounds directly within EarSketch, and import them from Freesound,[e] a large online collection of Creative Commons licensed sounds. This approach encourages students to identify musical genres and content that are personally meaningful to them and to incorporate this content into their own work.

An additional sidebar displays instructional materials for students, including text, runnable code examples, videos, multiple-choice questions, and slides. These are part of the EarSketch curriculum for Computer Science Principles.

Curriculum

The EarSketch curriculum is aligned with the programming standards of the College Board's Advanced Placement (AP) Computer Science Principles (CSP) course, as well as a related course that is a standard in the state of Georgia. AP CSP was launched in the fall of 2016 with a goal to offer a rigorous introduc-

e https://freesound.org

tory curriculum that broadens participation in computer science. The course introduces students to the creative aspects of programming, abstractions, algorithms, large datasets, the Internet, cybersecurity, and the impacts of computing across multiple domains.[4]

We aligned the EarSketch curriculum to CSP because of a shared goal of broadening participation in computing through a creative and authentic approach. CSP's curricular framework is broader than traditional computer science courses, with a focus on collaboration, analysis, communication, creativity, and connections to other disciplines. In contrast to other introductory computing courses, CSP is language agnostic. It does not mandate a specific programming language or problem domain: students submit performance tasks created with a programming language/environment of their choice, and they take a language-agnostic end-of-course exam. This all facilitates integrating EarSketch into the course.

The EarSketch curriculum for CSP consists of a ˜12-week module within the course that covers all of the CSP learning objectives related to programming and many of the objectives for creativity, abstraction, and algorithms. While we could have created a full-year CSP curriculum focused exclusively on music and EarSketch, we believe students should be exposed to multiple domains that are impacted by computing in addition to music; therefore, the EarSketch curriculum can readily be combined with curricula from Code.org[f] or Beauty and Joy of Computing[g] to implement a full-year CSP course.

The EarSketch CSP module is organized into three units. Each unit has an authentic challenge that requires the student to code musical concepts to satisfy the musical and technical criteria of the challenge. As an example, the first challenge requires the student to select a client that could be a business in their community or a school organization. The student must develop a 10- to 15-second musical introduction for a client advertisement that applies research shared with the student on how tempo and pitch affect mood. In addition, the student must apply musical effects, like volume fades or pitch shifts, to help create this mood.

Students share their music and code with their classmates and teacher to see if the intended mood is elicited and also discuss their code. Based on the constructive feedback they receive, they then iterate on their creation, share with their client to receive additional feedback, and further iterate to reach a final product. Students also use a rubric check sheet and write a justification of how their programming artifact fulfills the technical and artistic requirements of the project.

In open-ended projects such as this challenge, there is no single correct solution for an assignment. Students must collaborate and communicate with their classmates, their teacher, and external partners to iteratively refine the project goals, assess work in progress, and devise new musical and computational strategies to address feedback. The EarSketch CSP module follows this studio-based learning (SBL)[19] approach across all three units: designing an artifact; presenting work to peers and teachers, along with a detailed justification of the decisions made; discussing the work of peers and offering constructive questions and feedback; and revising work based on the feedback of others.

Many computing teachers are unfamiliar with this approach and are also new to teaching CSP and to the domain of music. We have thus developed scaffolding and support for teachers in three areas: teaching materials that include day-by-day lesson plans, slides, worksheets, mini-tasks, videos, project descriptions and rubrics, assessments, and integration guides; face-to-face and online professional development that introduces teachers to EarSketch, the curriculum, and these new pedagogical practices; and, a community where teachers can ask questions, share materials, and review additional training resources in both an online website and a series of in-person events.

Findings

We have measured pre-to-post changes in EarSketch students' attitudes toward computing, primarily in CSP courses. Like other CS educational interventions, EarSketch seeks to generate pre-to-post gains in students' CS content knowledge and has significantly done so across multiple research studies, with effect sizes that place learning in Hattie's[17] zone of 'typical teacher effects' and in the 'zone of desired effects.'

From interest to persistence in computing. EarSketch seeks not only to engage students in computing, but also to motivate students to persist in computing after the course. CSP teachers using EarSketch have compared it to other CS learning platforms and appreciate that EarSketch allows students to create artifacts interesting to students and to do so quickly. In an interview with our team, one teacher said:

"Well, we were doing [platform], if you're familiar with it, it's read, read, read, and do little things, but it wasn't real hands-on. ... So, when I put them on EarSketch, it was like, 'Whew!' They really got it. Where I've been teaching all these concepts that don't mean anything to you until you do them. So, EarSketch really implemented everything we'd kind of gone over up to that point and then some."

While building interest in computer science is important, teachers also describe the work they do to move students from initial interest to an intention to persist in further CS-related study. In particular, building interest among students with little initial interest or understanding of CS is challenging. A teacher whose class is comprised mainly of students who did not select the course shared:

"I tend to have students who are either placed in the course and have no idea what [CS] is, or just have absolutely no experience in programming. So, I definitely think EarSketch levels the playing ground. They're able to find something interesting about programming. They all like music. I think they thought it was fun, and I think it's engaging for them."

Another teacher explains that students who might not have persisted are signing up for AP Computer Science A (the course that typically follows CSP):

"As a result of using EarSketch, they're a lot more confident, and many of them have signed up for AP Computer Science [A] when they would not have before. Because now they feel like, 'Yeah, I can do this. I'm not afraid of

f https://studio.code.org/courses/csp-2018
g https://bjc.edc.org/bjc-r/course/bjc4nyc.html

programming. I'm not afraid of doing an actual language.'"

We have conducted several quantitative EarSketch studies that explore students' intention to persist in computing[11,24,29,38] through retrospective pre-post surveys. Three of the studies focused on high school students in introductory computing courses; the fourth study[38] focused on undergraduate students taking an introductory programming course to fulfill a computing requirement for non-majors.

Collectively, the studies included over 500 students in eight different institutions across four different academic years. Each of the four studies show statistically significant pre-to-post increases in intention to persist as well as in students' attitudes toward computing, typically with medium or large effect sizes. In Magerko et al.[24] female students expressed greater pre-to-post change across all attitudinal constructs and intention to persist than male students, with significantly greater gains in confidence, motivation, and identity. In that same study, a comparison of under-represented minority and majority students showed that both groups demonstrated significant increases in all attitudinal constructs and intention to persist, and that there was no significant difference between minority and majority student growth. In Siva et al.,[38] students in treatment sections of the course using EarSketch had significantly larger pre-to-post gains in intention to persist than students in comparison sections that did not use EarSketch.

We hypothesize that intention to persist may be activated by students' attitudes toward computing and that meaningful CS learning experiences might shift students' attitudes. Therefore, we explored the factors that contribute to students' increases in intention to persist. In McKlin et al.,[29] we conducted a path analysis to analyze student data in the context of this hypothesis. We found that students' identity as a computer scientist (such as, beliefs, expressions, and behaviors that motivate a person to align with or relate to a group) significantly predicts their intentions to persist in computing.

Authentic learning environment to foster identity. We theorize that EarSketch may support the growth of computing identity among students who typically do not pursue computing by providing a learning environment that students perceive to be thickly authentic.[15,23,37] One teacher offers:

"They can see the benefit of what they're learning, that real-world connection right away. I think that's what's so beneficial with using EarSketch, because they can see it."

Another teacher explains that EarSketch is meaningful to students because they are building music:

"[EarSketch] takes all that stuff they've learned and puts it into a hands-on audio, visual concept. It just makes so much sense to them once they hear it and see it. It's not just making that music play. It's 'how do I make that music play?' They got that day one."

In three of our studies,[11,29,38] we conducted a path analysis of student data to understand the relationship between perceived authenticity (called "Creativity Place" in Engelman et al.[11]), student attitudes, and intention to persist. In each study, authenticity consistently and significantly predicts positive changes in students' identity as a computer scientist along with positive changes in confidence, enjoyment, importance/usefulness, motivation, and personal creativity. While authenticity does not directly predict students' intention to persist, it does significantly predict attitudinal factors (such as identity) that in turn predict students' intention to persist.

Personal creativity. In our research, personal creativity includes characteristics of students who engage in a creative endeavor with computing and includes expressiveness, exploration, immersion, originality, sharing, and creative thinking skills. In our recent study examining the relationship between personal creativity and students' intention to persist in computing,[29] we found one aspect of personal creativity stands out: sharing. That is, sharing computing work among family and friends is more likely to predict students' intentions to persist in computing than any other factor of creativity.

A teacher explains how students share the technical and practical aspects of the work with family and friends:

"[They are] able to create something. It's theirs. They can show it to their friends. A lot of them like that aspect of it, that they're able to show them, 'Hey, I actually made this.' … They will say, 'Hey, this is what I made in EarSketch. This is similar to what you just made in GarageBand.'"

Ecosystem analysis. We have looked beyond student-level outcomes and have examined the school and classroom-level implementation of EarSketch. Because student learning is situated in a classroom and is affected not only by the teacher and students—but also by the school and district's infrastructure, capabilities, and culture—models of this larger ecosystem have enabled us to better understand what is needed to successfully implement and sustain EarSketch.

For example, based on observation data, our models captured a phenomenon known as the 'virtuous' and 'vicious' cycle of engagement with EarSketch. In the 'vicious' cycle, a student uses brute force repetition of more basic computational structures and avoids more algorithmic thinking, thus composing music without advancing computing knowledge. The 'virtuous' cycle, by comparison, is where students continue to develop both musical and computational skills side by side throughout the EarSketch course module. The models of these virtuous and vicious engagement cycles enabled the EarSketch team to address this common issue in creative computing learning platforms through a combination of new content in teacher professional development sessions and more scaffolding of student projects.[30]

Emerging Work

We have begun to cultivate communities of EarSketch students and teachers through both digital and physical means. We recently introduced sharing and collaboration tools to EarSketch for script sharing and collaborative editing, as well as live coding and time synchronization tools that help EarSketch users perform together.[36] We have staged EarSketch competitions to recognize exemplary projects and to help students discover the connections between algorithmic composition and music production, entrepreneurship, writing, and visual art.

Our team has also partnered with Northwestern University and the Museum of Science and Industry in Chicago to create a larger ecosystem of music plus computing tools that target informal educational settings, including a collaborative tabletop system for museums[40] and a tablet environment targeted for use in home and workshop settings.[13]

We have also begun to explore the potential of deep learning to tackle persistent design challenges in the creative computing education space. These tasks include auto-grading open-ended computing assignments (that is, using a large corpus of code to train a system to recognize evidence of specific content knowledge in student projects); providing students with real-time assistance in debugging code; and training co-creative learning companions that can provide both technical and creative ideas to students as they work.

Conclusion

In a recent focus group, a CSP student learning with EarSketch said:

"Before I thought coding was kind of like ... Not necessarily evil, but something that you pretty much had to do. ... So, I thought if maybe I knew a little bit I'd be something in the future. But now I actually want to do it, not because it will probably benefit me someday, but because it's also fun and engaging."

Her reflections exemplify the impact we hope to achieve with EarSketch. Students may choose to study computing because of the growing demand for these skills in the workforce, because of external pressure, or because they believe they are good at it. EarSketch, with its design that emphasizes dual-domain authenticity and immediate opportunity to act, offers a pathway for students to enjoy computing, to find it fun and engaging, and to want to pursue it simply because they love it.

Acknowledgments

EarSketch receives funding from the National Science Foundation (CNS #1138469, DRL #1417835, DUE #1504293, DRL #1612644, and IIP #1741045), the Scott Hudgens Family Foundation, the Arthur M. Blank Family Foundation, the Ruth L. Seigel Family Foundation, and the Google Inc. Fund of Tides Foundation. Any opinions, findings, and conclusions or recommendations expressed in these materials are those of the author(s) and do not necessarily reflect the views of the National Science Foundation or other funders. Many thanks to the members of the EarSketch research team—a full list is at http://earsketch.gatech.edu.

References

1. Aaron, S. et al. The development of Sonic Pi and its use in educational partnerships: Co-creating pedagogies for learning computer programming. *J. Music, Technology & Education 9*, 1 (2016), 75–94.
2. Abrams, D. Social identity on a national scale: Optimal distinctiveness and young people's self-expression through musical preference. *Group Processes & Intergroup Relations 12*, 3 (May 2009), 303–317; https://doi.org/10.1177/1368430209102841.
3. Ariza, C. Navigating the landscape of computer aided algorithmic composition systems: A definition, seven descriptors, and a lexicon of systems and research. In *Proceedings of the 2005 Intern. Computer Music Conference* (Barcelona, 2005).
4. Astrachan, O. et al. CS principles: Piloting a new course at national scale. In *Proceedings of the 42nd ACM Tech. Symp. on Computer Science Education.* ACM, New York, NY, USA, 2011, 397–398.
5. Bau, D. Droplet, a blocks-based editor for text code. *J. Computing Sciences in Colleges 30*, 6 (2015), 138–144.
6. Bau, D. et al. Pencil code: Block code for a text world. In *Proceedings of the 14th International Conference on Interaction Design and Children* (2015), 445–448.
7. Carroll, J. *Minimalism Beyond the Nurnberg Funnel.* MIT Press, Cambridge, MA, 1998.
8. Chusid, I. Beethoven-in-a-box: Raymond Scott's electronium. *Contemporary Music Review 18*, 3 (Jan. 1999), 9–14; https://doi.org/10.1080/07494469900640291.
9. Cope, D. *The Algorithmic Composer.* A-R Editions, Inc., 2000.
10. Detailed Race and Gender Information 2017; http://home.cc.gatech.edu/ice-gt/599.
11. Engelman, S. et al. Creativity in authentic STEAM education with EarSketch. In *Proceedings of the 2017 ACM SIGCSE Tech. Symp. on Computer Science Education.* ACM, New York, NY, 183–188.
12. Essl, K. Algorithmic composition. *The Cambridge Companion to Electronic Music.* N. Collins and J. d'Escrivan, eds. Cambridge University Press, 2007, 107–124.
13. Gorson, J. et al. TunePad: Computational thinking through sound composition. In *Proceedings of the 2017 Conference on Interaction Design and Children.* ACM, New York, NY, 484–489.
14. Guzdial, M. Teaching programming with music: An approach to teaching young students about logo. Logo Foundation, 1991.
15. Guzdial, M. and Tew, A.E. Imagineering inauthentic legitimate peripheral participation: An instructional design approach for motivating computing education. In *Proceedings of the 2nd Intern. Workshop on Computing Education Research.* ACM, New York, NY, 2006, 51–58.
16. Hamilton, R. et al. Social composition: Musical data systems for expressive mobile music. *Leonardo Music J. 21*, 1 (Nov. 2011), 57–64.
17. Hattie, J. *Visible Learning for Teachers: Maximizing Impact on Learning.* Routledge, 2012.
18. Hedges, S.A. Dice music in the 18th Century. *Music & Letters 59*, 2 (1978), 180–187.
19. Hendrix, D. et al. Implementing studio-based learning in CS2. In *Proceedings of the 41st ACM Tech. Symp. on Computer Science Education.* ACM, New York, NY, 2010, 505–509.
20. Hewner, M. and Knobelsdorf, M. Understanding computing stereotypes with self-categorization theory. In *Proceedings of the 8th Intern. Conference on Computing Education Research* (2008), 72–75.
21. Hiller, L.A.J. and Isaacson, L.M. *Experimental Music: Composition with an Electronic Computer.* McGraw-Hill, 1959.
22. Jaques, N. et al. Generating music by fine-tuning recurrent neural networks with reinforcement learning. In *Proceedings of Deep Reinforcement Learning Workshop, NIPS* (2016).
23. Lee, H.-S. and Butler, N. Making authentic science accessible to students. *International J. Science Education 25*, 8 (2003), 923–948.
24. Magerko, B. et al. EarSketch: A STEAM-based approach for underrepresented populations in high school computer science education. *ACM Trans. Computing Education 16*, 4 (2016).
25. Mahadevan, A. et al. EarSketch: Teaching computational music remixing in an online Web audio-based learning environment. In *Proceedings of the 1st Annual Web Audio Conference* (Paris, 2015).
26. Mahmoud, Q.H. Revitalizing computing science education. *IEEE Computer 38*, 5 (2005), 98–100.
27. Manaris, B. and Brown, A.R. *Making Music with Computers: Creative Programming in Python.* CRC Press, 2014.
28. McCartney, J. Rethinking the computer music language: SuperCollider. *Computer Music J. 26*, 4 (Dec. 2002), 61–68; https://doi.org/10.1162/014892602320991383.
29. McKlin, T. et al. Authenticity and personal creativity: How EarSketch affects student persistence. In *Proceedings of the 49th ACM Tech. Symp. on Computer Science Education.* ACM, New York, NY, 2018, 987–992.
30. Moore, R. et al. STEAM-based interventions in computer science: Understanding feedback loops in the classroom. In *Proceedings of the 2017 ASEE Annual Conference & Exposition.* (June 2017).
31. Peretz, I. and Zatorre, R.J. Brain organization for music processing. *Annual Review of Psychology 56*, (2005), 89–114.
32. Puckette, M. Combining event and signal processing in the MAX graphical programming environment. *Computer Music J.* (1991), 68–77.
33. Resnick, M. et al. Scratch: Programming for all. *Commun. ACM. 52*, 11 (Nov. 2009), 60–67.
34. Roads, C. *Microsound.* MIT Press, Cambridge, MA, 2004.
35. Ruthmann, A. et al. Teaching computational thinking through musical live coding in scratch. In *Proceedings of the 41st ACM Techn. Symp. on Computer Science Education* (2010), 351–355.
36. Sarwate, A. et al. Collaborative coding with music: Two case studies with EarSketch. In *Proceedings of the Web Audio Conference* (Berlin, 2018).
37. Shaffer, D.W. and Resnick, M. 'Thick' authenticity: New media and authentic learning. *J. Interactive Learning Research 10*, 2 (1999), 195–215.
38. Siva, S. et al. Using music to engage students in an introductory undergraduate programming course for non-majors. In *Proceedings of the 49th ACM Tech. Symp. on Computer Science Education.* ACM, New York, NY, 2018, 975–980.
39. Wing, J.M. Computational thinking. *Commun. ACM. 49*, 3 (Mar. 2006), 33–35.
40. Xambó, A. et al. Experience and ownership with a tangible computational music installation for informal learning. In *Proceedings of the 11th Intern. Conference on Tangible, Embedded, and Embodied Interaction.* ACM, New York, NY, 2017, 351–360.
41. Zölzer, U. *DAFX: Digital Audio Effects.* John Wiley & Sons, 2011.

Jason Freeman (jason.freeman@gatech.edu) is a professor and chair of the School of Music at Georgia Institute of Technology, Atlanta, GA, USA.

Brian Magerko (magerko@gatech.edu) is a professor and director of Graduate Studies in Digital Media at Georgia Institute of Technology, Atlanta, GA, USA.

Doug Edwards (doug.edwards@ceismc.gatech.edu) is a research associate for the Center for Education Integrating Science Mathematics and Computing at Georgia Institute of Technology, Atlanta, GA, USA.

Tom McKlin (tom@thefindingsgroup.org) is director at The Findings Group, Decatur, GA, USA.

Taneisha Lee (taneisha@thefindingsgroup.org) is lead evaluator at The Findings Group, Decatur, GA, USA.

Roxanne Moore (roxanne.moore@ceismc.gatech.edu) is a senior research engineer for the Center for Education Integrating Science Mathematics and Computing at Georgia Institute of Technology, Atlanta, GA, USA.

DOI:10.1145/3338537

Sketching the underlying system needed to facilitate metadata-private communication for several applications with a large user base.

BY YOSSI GILAD

Metadata-Private Communication for the 99%

ONLINE PRIVACY IS a prominent topic in the news and receives growing attention from the public. This motivated messaging services such as WhatsApp, Signal, and Telegram to deploy end-to-end encryption, which hides the content of messages from anyone who listens on the communication. While encryption is widely deployed, it does not hide metadata: anyone capable of tapping the network links can learn who is communicating with whom, at what times, and study their traffic volumes. Metadata reveals a lot about the underlying content. Public announcements by ex-government officials as well as the leaked Snowden documents have made it clear that intelligence organizations have a substantial interest in metadata even for encrypted communication since it often obviates the need for the actual content.[13,24,28]

The most popular system for hiding metadata—named Tor[7]—routes user traffic through a series of relay servers, as illustrated in Figure 1. In this fashion, the first relay only sees traffic to and from Alice, but it does not observe the other end of the conversation. Similarly, the last relay sees Bob's traffic, but does not observe the user at the other end. So even if just one of the relays is honest, that is, keeps secret which incoming message maps to what outgoing mes-

» key insights

- **The size of a user base is crucial for online privacy. If a metadata-private system only has a handful of users, then installing the system's client might raise suspicion. Therefore, to support a large user base and attract many users, metadata-private systems should be designed to be scalable and performant.**
- **Perfect security and privacy guarantees make metadata-private systems challenging to scale. It is possible to make trade-offs, providing weaker guarantees with a more scalable and performant system.**
- **The recent progress in metdata-private communication systems is impressive yet many challenges still exist, including supporting mobile devices and alleviating CPU bottlenecks.**

IMAGE BY MIREXON

sage that it forwards, the connection between Alice and Bob remains hidden. One of the key reasons for Tor's popularity is its performance, which can support mobile and desktop users and a variety of applications, such as messaging, Web surfing, and VoIP calls. However, Tor is vulnerable to attackers that can observe traffic going in and out of the honest relays. By tapping relays, attackers can correlate messages that a relay receives to those that it sends, and follow a message from its source to destination. In fact, it is sufficient to tap the first and last Tor relays to correlate traffic and break its privacy guarantees, as shown in Figure 2. Indeed, the Snowden documents revealed the U.S.'s National Security Agency (NSA) NSA and the U.K.'s Government Communications Headquarters (GCHQ) attempted to break the privacy provided by Tor in this manner (as well as by "contributing" their relays to the system).[24]

Recent research on systems that hide metadata focuses on dealing with a global adversary with the ability to monitor and inject network traffic on any link, comparable to nation-state organizations like the NSA or GCHQ. For example, various recent systems hide metadata for point-to-point messaging,[1,19,27,35,36] and others facilitate anonymous postings on public bulletin boards[3,18] in the presence of such an adversary.

Hiding metadata is complicated, and comes at a price. Metadata includes crucial information for functionality. As one notable example, the source and destination addresses, which identify the communication endpoints, are include d in every IP packet and are fundamental for establishing communication over the Internet. Other than explicit information recorded in network packets, a metadata private communication system also needs to obscure traffic correlations, such as the relation between the time at which messages were sent and the time they were delivered. If the attacker sees that Bob starts receiving messages when Alice starts sending messages and stops receiving messages when she stops sending them, then he can deduce that they are communicating even if the source and destination addresses in messages are hidden. Worse yet, the attacker might control Alice's link to the Internet, which allows him to perturbate her connection to trigger such events. For example, the attacker may drop some of Alice's messages and observe correlated reductions in throughput at Bob's end.[11,21]

As a result, many of the connections

Figure 1. Alice sends a message to Bob through Tor. The message routes through three relays to hide its metadata.

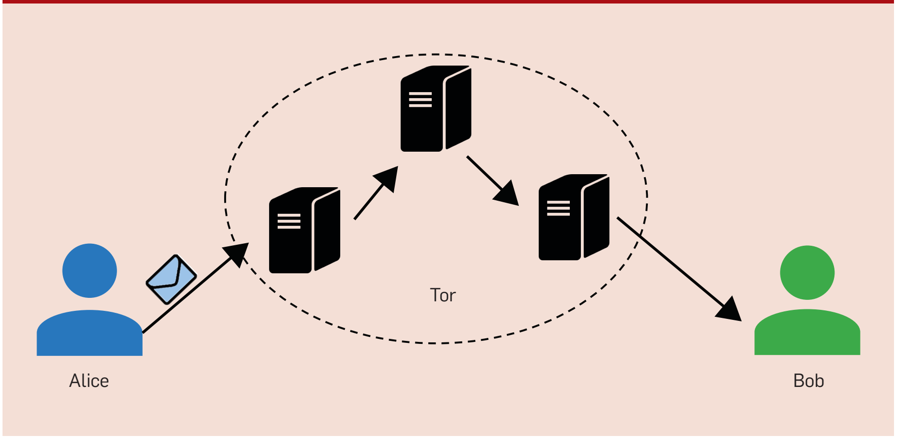
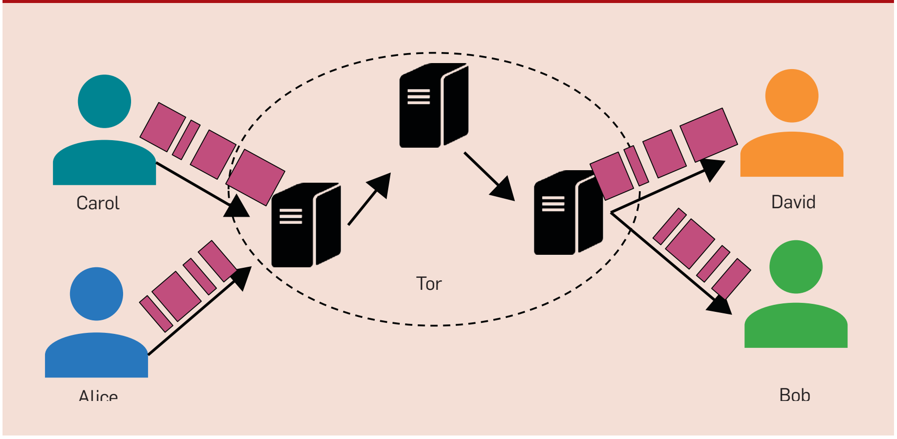


Figure 2. Eavesdropping on the communication links allows identifying which pairs of users communicate through Tor by correlating their traffic patterns.



through Tor are vulnerable to nation-state adversaries,[26] and the substantial interest of the intelligence community in metadata[13,28] means these attacks are a real problem in practice.

To hide metadata, the communication system needs to change the fundamentals that facilitate efficient communication over the Internet, such as packet routing and timely message delivery. The system might also need to constantly send messages, to hide when the user is actually communicating. These changes lead to significant challenges in designing practical metadata hiding communication systems and result in substantial overhead over the non-metadata-hiding "vanilla" counterparts. The overhead deems some applications, such as private voice and video chats, and some less-powerful devices like mobile phones, unusable with the current state of the art.

In this article explores the challenges ahead in hiding metadata to facilitate private communication through common types of applications. That is, allow two users who know one another to communicate without exposing that they are doing so. Hiding communication metadata can also facilitate anonymous communication, where users access a network service without revealing information about their identity. Some services refuse or throttle connections from anonymous users (for example, connected through Tor),[17,29] which dispirits adoption. The discussion here is focused on the simpler problem of facilitating private communication at large scale between users who want to communicate with one another. Informally, we ask: is it possible to hide metadata for popular types of applications for user-to-user communication? Some compromises in security and privacy must be made to support large-scale latency-sensitive applications. We motivate why, despite these compromises, supporting such applications with substantial privacy and security guarantees may be possible even with a large user base.

Dealing with Nation-State Adversaries

Designing systems that hide metadata in the face of resourceful organizations like nation-states requires dealing with several attack vectors: A nation-state might monitor traffic on network links, not just within its territory, but also at a global scale. It might also corrupt a substantial fraction of the system's servers in targeted attacks to read messages encrypted for these servers and find correlations between messages that they relay. Moreover, attacks are not confined to be technical. Nation-states might compel service providers to cooperate with them. For example, the NSA's PRISM program coerced public companies to disclose data that was not otherwise observable by the NSA.[12]

To understand where the performance penalties in dealing with these challenges stem from, we explain in more detail the attacks that metadata private communication systems need to resist and the defenses that these systems typically deploy.

Aside from the challenges listed here, which aim to break privacy, nation-states may also target the availability of a metadata-hiding service. They may block the service to stop users from communicating privately or even detain users who install the system's clients. We discuss why the key to dealing with such problems may lie in achieving good performance.

Traffic monitoring. IP addresses, service ports, message sizes and their transmission/receipt times are all directly observable to an attacker monitoring the traffic links. These observations reveal information about the underlying conversation, as discussed earlier. Traffic monitoring attacks are often also tough to detect since the attacker can remain passive, that is, avoid intervening with actual communication.

Internet routing increases the attack surface. Messages travel on the Internet through many hops and via several autonomous systems (independently administrated organizations). It is sufficient for an attacker to tap any of these links, or coerce any of the transient autonomous systems, to observe communication metadata. Internet routing gives end users very little control over the routes their messages take. They can choose their Internet service provider, but that provider selects the next hop (and that hop selects the next one). So forcing their messages to avoid routes through particular organizations is difficult. Worse yet, Internet routing is asymmetric, so messages traveling from Alice to Bob are likely to go over different links than those from Bob to Alice. Most protocols rely on bi-directional communication, which essentially doubles the attack surface,[31] as illustrated in Figure 3. In particular, any protocol that relies on the Internet's transmission control protocol (TCP) implicitly requires recipients to acknowledge every message they get, which makes 40% of connections through Tor vulnerable to nation-states that passively tap the network links in their territory.[26]

The underlying protocols that facilitate communication over the Internet were not designed to be secure. As a result, even if traffic would not typically route through an attacker-controlled organization, changing the traffic's path is usually feasible and with minimal effort. Mounting prefix hijacks only requires administrating one autonomous system and announcing someone else's address space. The false announcement manipulates the Internet's routing protocol into delivering to the attacker messages intended to the (hijacked) address space. In fact, Internet routing is so fragile that it is difficult to distinguish between common configuration mistakes and deliberate attacks, providing attackers a reasonable excuse even if they are detected. In 2013, an ISP in Belarus hijacked Internet traffic intended for Iceland.[4] In a similar, but believed to be accidental, incident in 2014 Indosat hijacked traffic for 44 Tor relays (as well as other destinations).[40]

Dealing with traffic monitoring. To mitigate traffic monitoring attacks, it is crucial to ensure traffic patterns appear the same no matter which users communicate and regardless of the time, length, and content of their conversation. To this aim, many systems that hide metadata route fixed-size messages through a relay server, which unlinks input messages from output messages.

The relay server shuffles the output messages' transmission order. Since the shuffle permutation is secret, the adversary cannot map a message output from the relay back to the corresponding input. (Hop-to-hop encryption between the users and the relay unlinks the contents of the relay's inputs from its outputs.) In this manner, directly observable fields in the message, like source and destination addresses, do not reveal information about the user-to-user communication metadata. To prevent timing correlations, many systems are synchronous. They operate in rounds, where at the beginning of the round each user submits a message, and at the end of the round, the system delivers the message to its destination. Synchronicity allows dealing with timing attacks since all message exchanges happen on round boundaries.

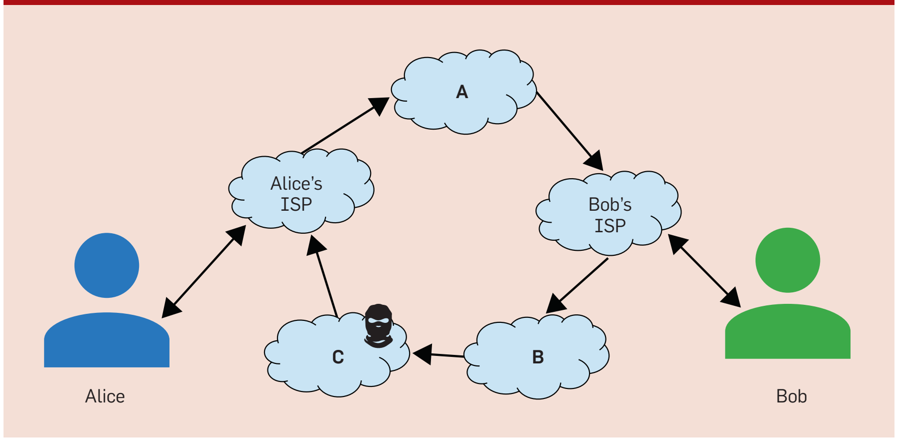
Routing through a relay often forces messages through unduly long routes, and synchronicity implies that a relay cannot forward even a single message before it is done processing the entire batch of messages for a particular round. Another unfortunate consequence of synchronous designs is that they require all users to keep submitting messages to the system at every round or it becomes apparent when users are involved in a conversation. Short rounds would facilitate low latency communication, but require everyone to send messages all of the time. Having clients constantly send messages increases the load on the system and the cost for the clients. It implies that operating a client has high network and energy costs that make existing synchronous solutions prohibitively expensive to run on mobile devices, which are limited by data plans and battery life. Support for mobile devices in a synchronous system remains a largely unsolved challenge, which I discuss later.

Indeed, asynchronous systems are more perfomant than others[7,27] and better accommodate latency-sensitive applications and mobile clients, but may suffer from statistical attacks that correlate users' traffic patterns.[7,27]

Corrupt servers and colluding operators. Attackers may compromise the system's servers. Moreover, nation-states might be in a position to coerce a server operator to cooperate with them.[12] Such attacks would expose the server's secrets to the attacker. In the typical relay-based operation sketched here, the server's secrets would allow the attacker to learn the message-shuffle permutation and map messages sent from the relay server back to its inputs, thereby unveiling the source and destination of each message. One standard approach for resisting malicious servers is to route each message through several relays, each administered by a different organization, as shown for Tor in Figure 1.

A typical design goal is to guarantee that if any of these servers are honest, metadata remains hidden. Of course, processing a message by multiple servers induces latency, adding to the challenge of supporting latency-sensitive applications in practice. Notably, it is possible to avoid trusting any of the system's servers using sophisticated cryptographic constructs such as private information retrieval, which obviates routing messages through multiple relays.

Figure 3. The attacker can only observe traffic going through autonomous system C, which allows him to monitor traffic from Bob to Alice and learn they are communicating.



Service blocking. An attacker might give up on breaking the system's privacy guarantees, and block connections to the system altogether or detain its users. In particular, there is evidence that some governments fingerprint and block connections to Tor.[2,38] One approach for protecting against fingerprinting is to disguise communication as other popular services, which are already perceived legitimate.[23,22,27] However, disguising the service requires keeping its server addresses hidden to avoid blacklisting (see discussion in Moghaddam[23]). The approach taken by the Tor project to protect against blacklisting is to secretly distribute ephemeral addresses of "bridge" nodes, relay servers to which users directly connect to access Tor. Bridges ultimately get discovered and blocked, creating a cat-and-mouse game between Tor's operators and some governments.

A perhaps more concerning problem is that attackers might consider running the system's client to be suspicious in its own right. It seems that the most effective way to combat this concern is to make the metadata-hiding service so popular, such that using it does not raise suspicion (an argument originally made by Tor's designers[6]). Making a privacy-preserving service appeal to a broad audience of users, who often do not worry much about their privacy, requires achieving comparable performance to the available non-metadata-hiding alternative. Achieving good performance together with the design constraints forced by the privacy or security requirements is difficult.

Online Privacy at Scale

A large user base is crucial for privacy; so metadata-hiding communication systems must be designed to scale. One standard approach to scaling a system, in general, is to design it such that the more servers are available, the more users it can support. In the context of metadata-hiding systems, which typically rely on volunteer organizations to contribute and operate servers, we would like the performance to improve as the number of those organizations grows. This property is known as horizontal scaling. (Contrasted to vertical scaling, which requires every provider to contribute beefier machines and more bandwidth.) Recent research shows a fundamental trade-off:[5] to achieve low-latency communication without compromising on privacy, the system must increase its bandwidth proportionally to the number of users; so horizontal scaling should be treated as a first-class design principle, as it allows to keep the load on each contributing organization moderate even when the user base grows large. Through horizontal scaling, Tor can serve millions of concurrent users. It distributes the client-load across more than 6,000 servers and reaches acceptable latency to support a variety of Internet applications.[a]

a See https://metrics.torproject.org for measurements about Tor's deployment.

The research community's efforts to build metadata-private systems that can resist nation-state adversaries have so far resulted in systems that provide medium to high latency, ranging from several seconds to hours. Three recent examples, Atom, Stadium, and Karaoke,[18,19,35] have shown how to design such communication systems that scale horizontally. Notably, with 100 organizations contributing servers, Karaoke can exchange messages between two million users every seven seconds—over 20 billion messages per day.[19] Exchanging this many messages per day falls in the ballpark of popular (encrypted, but not metadata-hiding) messaging applications such as WhatsApp and Telegram, which reported delivering 55 billion and 15 billion messages per day.[3,24] However, relatively high latency has limited the applications for metadata-private systems. For example, it may be acceptable for an email message to reach the recipient's mailbox with a latency of a few minutes, but messaging applications are expected to deliver within seconds, and voice and video chats require sub-second latency for adequate user experience that is key for broad adoption. This limitation is unfortunate since latency-sensitive communication mediums, such as VoIP calls and video conferencing, are popular. For example, in 2013, Microsoft reported two billion minutes of Skype calls every day.[22]

The primary challenge in designing communication systems that hide metadata is achieving a combination of three crucial goals: providing strong privacy guarantees, resisting attacks from nation-state adversaries, and attaining sufficient performance to support a target higher-level application (such as email, text messaging, or voice chats) with the adequate user experience.

The cost of perfect guarantees. Targeting the best privacy guarantees under the most challenging trust assumptions leads to performance problems. Specifically, challenging the ability of the design to scale to support a large user base.

Privacy. It is possible to design systems that avoid leaking any information at the cost of excessive communication or computation. For example,

Table 1. The 3-way trade-off in state-of-the-art systems. More performant systems provide weaker privacy and security guarantees. (Karaoke and Pung are horizontally scalable and evaluated with 100 servers.)

↑ Better Performance — Better Privacy and Security ↓

System	Privacy	Trust Assumption	Latency (At 2M Clients)	Throughput	Horizontally Scalable?
Tor[7]	vulnerable	one out of client-selected servers is honest	sub-second	very high (millions of users' Web traffic)	✓
Karaoke[19]	differential privacy	80% honest	7 sec	571k msgs/sec	✓
Vuvuzela[36]	differential privacy	any-trust	1 min	50k msgs/sec	×
Pung[1]	no metadata leak	zero-trust	18 min	2k msgs/sec	✓

a system where each user sends every message to all others can resist passive and active attacks.[3] Cryptographic techniques like DC-nets[39] and Private Information Retrieval,[1] keep the client's communication cost to a small constant, but introduce computational overheads that are quadratic in the number of users. These designs[1,3,39] provide the strongest form of privacy that users could hope for, but due to the overhead of these schemes performance suffers severely as the userbase grows.

However, even if the system facilitates communication without leaking any information about metadata, some information might leak just by the limitations of a human user. For example, if Alice is currently on a voice call with Bob, she is probably incapable of simultaneously talking with another user. Therefore, if the attacker tries to call Alice and Bob at the same time and the two do not respond, then he learns some statistical information about the possibility that they are communicating. By repeating this experiment, the attacker might find that Alice and Bob's availability is correlated, and reach an informed conclusion about whether they communicate. A fundamental question in designing metadata-hiding systems is therefore what kind of information leakage is acceptable for the particular application, and whether we can trade some leakage to get better performance.

Strength of trust assumptions. Systems must have clear underlying trust assumptions to provide meaningful privacy guarantees. Weaker assumptions mean the system's privacy guarantees are more likely to hold in practice. The weakest form of trust assumption—referred to as zero-trust—is not trusting the system at all. It means the system's servers facilitate communication, but even if they all turn out to be malicious, the system maintains its privacy guarantee—to keep communication metadata hidden (although malicious servers might still prevent users from communicating).

The zero-trust assumption inevitably comes with a significant computational cost. To understand why, consider an idealized scenario where users deposit messages at a server, and each user can poll that server for messages intended for them; fetching a message without leaking information about which one is being fetched can be done using cryptographic protocols for private information retrieval. Fundamentally, however, the server must process all deposited messages for each user query to be untrusted. Otherwise, the server must know that some user could not have sent a message to some other user, so some information about the metadata was surely exposed to the server. This implicit computational requirement leads to significant challenges in supporting many users. As one concrete example, Pung[1] uses zero-trust as its underlying security assumption, with 2M users its communication latency grows to 18 minutes.

Internet routing is so fragile that it is difficult to distinguish between common configuration mistakes and deliberate attacks, providing attackers a reasonable excuse even if they are detected.

The 3-way trade-off. To get around the performance challenges discussed earlier, it seems necessary to compromise on weaker privacy guarantees and stronger trust assumptions. Here, possible compromises are examined as well as what they enable in terms of performance. The accompanying table summarizes the trade-off points of several recent proposals.

Privacy vs. performance. Hiding all communication metadata, no matter what actions the attacker takes, results in significant overhead. However, as a recent design shows, it is possible to efficiently prevent any leakage of information about a conversation's metadata when the attacker is passive.[19] This is important because, as we noted earlier, passive attacks are tough to detect.

It is also possible to avoid severe performance penalties in case the attacker is active by relaxing the privacy goal to allow leaking a small amount of statistical information on every message exchange. Systematic mechanisms for quantifying and limiting information leakage to an adversary were studied through differential privacy.[8] Differentially private systems protect an individual's data by stochastically noising the system's outputs which the attacker can observe. These systems provide a weaker notion of privacy than those that rely on message broadcast or private information retrieval, and have seen significant adoption in practice (for example, by Google[9] and Apple[14]).

In context of communication systems, differential privacy limits information leakage by adding dummy messages as cover traffic.[19,20,35,36] The system's servers generate these dummies, and their amount is decided at random by each server in every communication round according to a fixed distribution (set by the system's designers). The stochastic process of adding dummy messages to user messages limits what the attacker could learn on the actual user-to-user communication from whatever attack he executes. The more dummy messages the system processes, the less information that might leak on each message-exchange. The performance benefit of differential privacy stems from the fact that the quantity of dummies is independent of the number of users and messages they exchange.

Strength of trust assumptions vs. performance. Zero trust induces high computational costs, which limits the ability to support a large user base. It is therefore important to identify weaker trust assumptions that are likely to hold in practice. Two other forms of trust assumptions are common.

Any-trust. One common alternative is to deploy the communication service over several servers administered by independent organizations, and to trust that at least one of these organizations is honest (without requiring users to trust a specific one). Distributing trust across many servers in the context of metadata-private communication systems dates back to Chaum's mixnets in the 1980s. By relaxing the security goal from zero-trust to any-trust, we can avoid the overhead of scanning all senders' inputs for each output that the system provides to a recipient. In practice, Vuvuzela,[36] the most performant any-trust system, can exchange messages between 2M users in about a minute (over an order of magnitude of improvement in latency and throughput over Pung).

A minute of latency is, however, far from being on par with vanilla messaging applications. A key reason for this performance handicap is that the any-trust assumption excludes horizontal scaling. Since under this assumption there may be just one honest server in the entire system, each server must process all messages (otherwise some messages might have skipped processing by the honest server, leaving their metadata exposed). In contrast, vanilla applications distribute the load over many servers.

A fraction of the system's servers are honest. It seems inevitable to require a stronger trust assumption than zero- or any-trust to be able to scale the system to support a large user base. A much more performance-friendly assumption is that some fraction of the system's servers are honest. It allows sharding the load of user messages among servers (enabling horizontal scaling).[18,19,35] Under this assumption, a user's message may be processed only by a small subset of the servers, where one server in the subset is assumed to be honest (that is, each subgroup of servers is any-trust). Karaoke, which operates under this assumption, improved on Vuvuzela's performance by about an order of magnitude in latency and throughput.[19]

Discussion

We have so far discussed the difficulties in building popular means of communication that hide metadata. However, recent progress in research on metadata-hiding systems is promising. We next motivate why building such systems, that support low-latency applications like voice and video chats, may be tangible and discuss some of the remaining challenges that the research community would need to tackle to make it happen.

Metadata-private low-latency communication at scale is tangible. The latency of communication through today's systems is high, but there is room for optimism about supporting low-latency applications in the future. This section explores why providing meaningful metadata privacy guarantees for important types of applications may be feasible.

Focusing on human communication. The volume of machine-to-machine communication proliferates, and constant improvements in network infrastructure support this growth. Latency-sensitive human-to-human communication typically involves lively interactions between people. As such, users are typically not part of simultaneous conversations and the duration of conversations is often limited. For many types of applications, such as voice calls, these properties imply the number of conversations and the amount of metadata we need to protect does not grow as rapidly as machine-to-machine communication (as one example, the average monthly mobile voice minutes per person in the U.K. increased only by 10% between 2008 and 2013[30]). The difference in growth rates suggests that systems' capabilities may catch-up with the amount of human-generated communication, and its metadata that we might wish to hide.

Leaking some information may be acceptable. As discussed previously, even a perfect metadata-hiding communication system might not prevent the attacker from learning some information about the communication (for example, since a user might only be able to have one conversation at a time). The challenge, therefore, lies in leaking as little information as possible while providing solid performance. Differential privacy seems to be a promising direction in consolidating this trade-off. It allows to limit and quantify the amount of information the system leaks to an attacker and allows to circumvent the computation and communication overheads of other approaches (where the system leaks no additional information about its users' traffic).

Challenges ahead. Recent works show that metadata-private systems can provide meaningful privacy guarantees and support a large user base. In the last eight years, the throughput of metadata-private systems has increased from a few hundreds of messages per second[39] to over 570k messages per second.[19] Achieving this improvement is accredited to theoretical advances in cryptographic constructs and differential privacy, horizontally scalable designs, and engineering efforts. However, despite these advances, the current state-of-the-art metadata-private communication systems still induce significant performance penalties. Here are three remaining challenges that current systems face, which, if alleviated, could dramatically improve performance and drive user adoption:

Supporting mobile devices. A common approach to avoid exposing correlations between the times that Alice and Bob are online and might be communicating is to have the clients regularly send messages (sending dummies when the users are not involved in a conversation). This approach, which is taken by state-of-the-art systems, induces significant cost to battery life and data-plans when running on mobiles. To support mobile clients used by a massive portion of the Internet's users, it seems necessary to rethink this strategy.

Reducing computations. The recent horizontally scalable designs distribute the communication overhead over many contributing organizations. The state-of-the-art systems, however, make extensive use of public key cryptography and rely on hefty cryptographic protocols like zero-knowledge proofs of shuffle and correct decryption. In particular, the performance of the horizontally scalable systems in the table—Karaoke and Pung—is bounded by computations (see experimental evaluation.[1,19]). Finding a way to minimize the use of these cryptographic constructs, such as by establishing persistent private sessions and using symmetric-key cryptography within these sessions (as often done when hiding content), would alleviate the computational bottleneck and reduce communication latency significantly.

Improving the topology. It is common to route messages through servers operated by organizations in different political and geographic regions, to reduce the chance that the organizations administrating these servers would collude or coerced to expose secrets. Topology studies were performed mostly on Tor, to avoid routing through specific autonomous systems[16,25] (combating the attacks mentioned previously) and to avoid overloading specific relays.[15]

A largely remaining challenge is to optimize the route that messages would take through different geographic regions, so as to avoid sending messages through an overly long distance. A route with many distant randomly selected hops[18,19,35,36] means that even if each server only relays a small portion of the messages, and does not perform any computationally heavy processing, the aggregate of the interserver latencies might be too expensive to support some applications. A significant challenge in supporting latency-sensitive applications is therefore to identify better routing topologies, which allow to mix all messages for privacy, yet do not require messages to go through many hops for performance.

Acknowledgments
The author thanks Sharon Goldberg, David Lazar, Michael Schapira, Adam Suhl, Moshe Vardi, Nickolai Zeldovich, and the anonymous reviewers for their helpful comments on earlier versions of this article.

References
1. Angel, S. and T. V. Setty, S. Unobservable communication over fully untrusted infrastructure. In *Proceedings of OSDI*, 2016. K. Keeton and T. (Eds.). USENIX Assoc., 551–569.
2. Aryan, S., Aryan, H. and Halderman, J.A. Internet censorship in Iran: A first look. In *Proceedings of FOCI*, 2013. J.R. Crandall and J. Wright (Eds.). USENIX Assoc.
3. Corrigan-Gibbs, H., Boneh, D. and Mazières, D. Riposte: An anonymous messaging system handling millions of users. In *Proceedings of IEEE Symposium on Security and Privacy*, 2015. IEEE Computer Society, 321–338; http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=7160813
4. Cowie, J. New Threat: Targeted Internet Traffic Misdirection; http://www.renesys.com/2013/11/mitm-internet-hijacking.
5. Das, D., Meiser, S., Mohammadi, E. and Kate, A. Anonymity trilemma: Strong anonymity, low bandwidth, low latency—Choose two. In *Proceedings of Security and Privacy*. IEEE, 2018.
6. Dingledine, R. and Mathewson, N. Anonymity loves company: Usability and the network effect. In *Proceedings of Workshop on the Economics of Information Security*, 2006.
7. Dingledine, R., Mathewson, N. and Syverson, P.F. Tor: The second-generation onion router. In *Proceedings of USENIX Security Symposium*, 2004. M. Blaze (Ed.). USENIX, 303–320.
8. Dwork, C., McSherry, F., Nissim, K. and Smith, A.D. Calibrating noise to sensitivity in private data analysis. *TCC 3876* (2006). Springer, 265–284.
9. Erlingsson, U., Pihur, V., and Korolova, A. RAPPOR: Randomized aggregatable privacy-preserving ordinal response. In *Proceedings of ACM Conf. Computer and Communications Security*, 2014. G.-J. Ahn, M. Yung, and N. Li (Eds.). ACM, 1054–1067; http://dl.acm.org/citation.cfm?id=2660267
10. Finley, K. Half of the Internet is now encrypted. This makes everyone safer; https://www.wired.com/2017/01/half-web-now-encrypted-makeseveryone-safer/.
11. Gilad, Y. and Herzberg, A. Spying in the dark: TCP and Tor traffic analysis. *Privacy Enhancing Technologies, LNCS* 7384 (2012). S. Fischer-Hübner and M.K. Wright (Eds.). Springer, 100–119.
12. Greenwald, G. and MacAskill, E. NSA Prism program taps in to user data of Apple, Google and others; https://www.theguardian.com/world/2013/jun/06/us-tech-giants-nsa-data.
13. Hayden, M. The price of privacy: Re-evaluating the NSA. *Proceedings of the Johns Hopkins Foreign Affairs Symposium.* (Apr. 2014); https://www.youtube.com/watch?v=kV2HDM86XgI&t=17m50s.
14. Apple, Inc. Differential Privacy, 2016; https://www.apple.com/privacy/docs/Differential_Privacy_Overview.pdf.
15. Johnson, A., Jansen, R., Hopper, N., Segal, A. and Syverson, P. PeerFlow: Secure load balancing in Tor. *Proceedings of PoPETs 2* (2017), 74–94.
16. Johnson, A., Wacek, C., Jansen, R., Sherr, M. and Syverson, P. Users get routed: Traffic correlation on Tor by realistic adversaries. In *Proceedings of ACM Conf. Computer and Communications Security*, 2013 A-R Sadeghi, V.D. Gligor and M. Yung (Eds.). ACM, 337–348; http://dl.acm.org/citation.cfm?id=2541806
17. Khattak, S. et al. Do you see what I see? Differential treatment of anonymous users. In *Proceedings of NDSS*, 2016. The Internet Society; https://bit.ly/2XbOnZ6
18. Kwon, A., Corrigan-Gibbs, H., Devadas, S. and Ford, B. Atom: Horizontally scaling strong anonymity. In *Proceedings of SOSP*, 2017. ACM, 406–422; http://dl.acm.org/citation.cfm?id=3132747
19. Lazar, D., Gilad, Y. and Zeldovich, N. Karaoke: Fast and strong metadata privacy with low noise. In *Proceedings of OSDI*, 2018. USENIX Assoc.
20. Lazar, D. and Zeldovich, N. Alpenhorn: Bootstrapping secure communication without leaking metadata. In *Proceedings of OSDI*, 2016. K. Keeton and T. Roscoe (Eds.). USENIX Assoc., 571–586.
21. Levine, B.N., Reiter, M.K., Wang, C. and Wright, M.K. 2004. Timing attacks in low-latency mix systems (extended abstract). *Financial Cryptography LNCS*, 3110. A. Juels (Ed.). Springer, 251–265.
22. Microsoft. 2 Billion Minutes a Day! Skype blog; https://bit.ly/2EGMYm2
23. Moghaddam, H.M., Li, B., Derakhshani, M., and Goldberg, I. SkypeMorph: Protocol obfuscation for Tor bridges. In *Proceedings of ACM Conf. Computer and Communications Security*. T. Yu, G. Danezis, and V.D. Gligor (Eds.). ACM, 97–108; http://dl.acm.org/citation.cfm?id=2382196
24. National Security Agency. Tor stinks. *The Guardian.* (Oct. 2013); https://bit.ly/2Qzntb7.
25. Nithyanand, R., Singh, R., Cho, S. and Gill, P. Holding all the ASes: Identifying and circumventing the pitfalls of AS-aware Tor client design. CoRR, 2016; http://arxiv.org/abs/1605.03596
26. Nithyanand, R., Starov, O., Gill, P., Zair, A. and Schapira, M. Measuring and mitigating AS-level adversaries against Tor. In *Proceedings of NDSS*, 2016. The Internet Society; https://bit.ly/2wqK54o
27. Piotrowska, A.M., Hayes, J., Elahi, T., Meiser, S. and Danezis, G. The Loopix anonymity system. In *Proceedings of USENIX Security Symposium*, 2017. E. Kirda and T. Ristenpart (Eds.). USENIX Assoc., 1199–1216.
28. Rusbridger, A. The Snowden leaks and the public; https://www.nybooks.com/articles/2013/11/21/snowden-leaks-and-public/
29. Singh, R. et al. Characterizing the nature and dynamics of Tor exit blocking. In *Proceedings of USENIX Security Symposium*, 2017. E. Kirda and T. Ristenpart, (Eds.). USENIX Assoc., 325–341.
30. Statistica, the statistics portal. Average monthly outbound mobile voice minutes per person in the United Kingdom (UK) from 2008 to 2013 (in minutes). (2013).
31. Sun, Y. et al. RAPTOR: Routing attacks on privacy in Tor. In *Proceedings of USENIX Security Symposium*, 2015. J. Jung and T. Holz (Eds.). USENIX Assoc., 271–286.
32. Telegram. 15 Billion Telegrams Delivered Daily. Telegram announcement, 2017; https://telegram.org/blog/15-billion.
33. The Tor project. Pluggable Transports, 2017; https://www.torproject.org/docs/pluggable-transports.
34. Tung, L. WhatsApp: Now one billion people send 55 billion messages per day, 2017; http://www.zdnet.com/article/whatsapp-now-one-billion-people-send-55-billion-messages-per-day/.
35. Tyagi, N., Gilad, Y., Leung, E., Zaharia, M. and Zeldovich, N. Stadium: A distributed metadata-private messaging system. In *Proceedings of SOSP*, 2017. ACM, 423–440; http://dl.acm.org/citation.cfm?id=3132747
36. Hooff, J., Lazar, D., Zaharia, M. and Zeldovich, N. Vuvuzela: Scalable private messaging resistant to traffic analysis. In *Proceedings of SOSP*, 2015. E.L. Miller and S. Hand (Eds.). ACM, 137–152; http://dl.acm.org/citation.cfm?id=2815400
37. Weinberg, Z. et al. StegoTorus: A camouflage proxy for the Tor anonymity system. In *Proceedings of ACM Conf. on Computer and Communications Security*, 2012. T. Yu, G. Danezis and V.D. Gligor (Eds.). ACM, 109–120; http://dl.acm.org/citation.cfm?id=2382196
38. Winter, P. and Lindskog, S. How the great firewall of China is blocking Tor. In *Proceedings of FOCI*, 2012. R. Dingledine and J. Wright (Eds.). USENIX Assoc.
39. Wolinsky, D.I., Corrigan-Gibbs, H., Ford, B. and Johnson, A. dissent in numbers: Making strong anonymity scale. In *Proceedings of OSDI*, 2012. C. Thekkath and A. Vahdat (Eds.). USENIX Assoc., 179–182.
40. Zmijewski, E. Indonesia Hijacks the World, 2013; https://dyn.com/blog/indonesia-hijacks-world/.

Yossi Gilad is Senior Lecturer at the Hebrew University of Jerusalem, Israel.

ACM Transactions on
Data Science

P. 96
Technical Perspective
From Virtual Worlds to Digital Fabrication
By Sylvain Lefebvre

P. 97
OpenFab: A Programmable Pipeline for Multimaterial Fabrication
By Kiril Vidimče, Szu-Po Wang, Jonathan Ragan-Kelley, and Wojciech Matusik

DOI:10.1145/3344806

To view the accompanying paper, visit doi.acm.org/10.1145/3344808

Technical Perspective From Virtual Worlds to Digital Fabrication

By Sylvain Lefebvre

COMPUTER GRAPHICS HAS enabled game developers to create vast open worlds full of wonders and dangers that players routinely explore and interact with. These worlds are not made of physical matter; instead, they are defined by a surface geometry—often triangles—and material parameters, such as color and reflectance. Rendering algorithms use this information to form a rapid sequence of images on screen, simulating the viewpoint of a moving virtual observer.

This ability to depict large open worlds has been a fundamental challenge from the infancy of computer graphics. The problem is twofold: describing the virtual world in all its intricate details, and being able to store and process this data for rendering. Describing a small virtual scene like a single room is not a problem: a skilled artist can use CAD software to model the surface geometry of each object, as well as specify the color of its parts. However, this quickly becomes impractical as the virtual scene grows in extent to become, for instance, an entire planet. Similarly, even if an entire planet could be modeled, the data would simply not fit the computer memory.

For these reasons, the field of computer graphics has focused intensely on dealing with very large amounts of geometry and material information. This has had a deep influence on the software, hardware, and industrial practices in our field. One key idea, pioneered by researchers such as Ken Perlin and Kenton Musgrave, is the notion of *proceduralism*: The idea that detailed geometric and material information does not have to entirely exist in memory. Instead, it can be generated on-the-fly, when needed, for rendering a single viewpoint. This is a powerful idea: The screen onto which an image is displayed has a limited resolution, and the amount of data visible in a single viewpoint is only a tiny subset of what could be the entire universe. By computing, or rather *synthesizing* only the content required for the current view, it becomes possible to explore worlds without bounds: more content is synthesized as the user wanders deeper into the virtual landscape. The content is then described by procedures, small algorithms that generate details whenever required from a simpler description of the scene.

How does this relate to additive manufacturing? As noted in the following paper by Vidimče et al., the rapid increase in both print resolution and print volumes, combined with the ability to mix different materials, leads to a very similar situation. Describing a 3D model in all its intricate details becomes rapidly infeasible. The challenge is not only in describing the object using available tools, but also in being able to store and process this description before fabrication. It might seem surprising: this is, after all, a single object. However, 3D printing requires specifying the material *at every point in the volume*, while most often virtual worlds require only describing surfaces. In addition, the print resolution is increasing toward micron accuracy. As a consequence, the number of points that must be specified to fully exploit the capabilities of the 3D printers grows very rapidly.

3D printing requires specifying the material *at every point in the volume*, while most often virtual worlds require only describing surfaces.

Additive manufacturing technologies fabricate an object layer after layer, from bottom to top. Each layer can be thought of as a two-dimensional grid of little cubes, where each cube is either empty, or will be filled with one or a mixture of materials. Taken together, the layers form a large three-dimensional grid of cubes, called *voxels*. Even with today's limitations, a print using the full extent of the printer can have billions of voxels. Fortunately, when the part is being fabricated, only a single layer needs to be in memory; this is akin to the limited viewpoint of the virtual observer in a virtual world. Thus, the same methodologies apply—rather than storing the object in all its intricate details, the details can be synthesized only when needed by the fabrication process, layer by layer.

The authors of OpenFab propose to revisit the processing pipeline that turns a 3D model into machine instructions in light of the solutions developed in computer graphics. In particular, rather than specifying the object by handpicking a material in each of its voxels, users can write small algorithms that synthesize the content when it is needed for fabrication. This integrates in an elegant pipeline that affords for unprecedented design flexibility, while simultaneously answering computational challenges. Suddenly, it becomes possible to fully exploit the high-resolution capabilities of the additive manufacturing processes. This unlocks a vast number of possibilities, from aesthetics to novel optical and mechanical properties, triggered by micro-structures embedded within the object's volume. C

Sylvain Lefebvre (sylvain.lefebvre@inria.fr) is a senior researcher at Inria, Nancy–Grand Est, France.

DOI:10.1145/3344808

OpenFab: A Programmable Pipeline for Multimaterial Fabrication

By Kiril Vidimče, Szu-Po Wang, Jonathan Ragan-Kelley, and Wojciech Matusik

Abstract

3D printing hardware is rapidly scaling up to output continuous mixtures of multiple materials at increasing resolution over ever larger print volumes. This poses an enormous computational challenge: large high-resolution prints comprise trillions of voxels and petabytes of data, and modeling and describing the input with spatially varying material mixtures at this scale are simply challenging. Existing 3D printing software is insufficient; in particular, most software is designed to support only a few million primitives, with discrete material choices per object. We present OpenFab, a programmable pipeline for synthesis of multimaterial 3D printed objects that is inspired by RenderMan and modern GPU pipelines. The pipeline supports procedural evaluation of geometric detail and material composition, using shader-like *fablets*, allowing models to be specified easily and efficiently. The pipeline is implemented in a streaming fashion: only a small fraction of the final volume is stored in memory, and output is fed to the printer with a little startup delay. We demonstrate it on a variety of multimaterial objects.

1. INTRODUCTION

State-of-the-art 3D printing hardware is capable of mixing many materials at up to 100s of dots per inch resolution, using technologies such as photopolymer phase-change inkjet technology. Each layer of the model is ultimately fed to the printer as a full-resolution bitmap where each "pixel" specifies a single material and all layers together define on the order of 10^8 voxels per cubic inch. This poses an enormous computational challenge as the resulting data is far too large to directly precompute and store; a single cubic foot at this resolution requires at least 10^{11} voxels and terabytes of storage. Even for small objects, the computation, memory, and storage demands are large.

At the same time, it is challenging for users to specify continuous multimaterial mixtures at high resolution. Current printer software is designed to process polygon meshes with a single material per object. This makes it impossible to provide a continuous gradation between multiple materials, an important capability of the underlying printer hardware that is essential to many advanced multimaterial applications (e.g., Wang et al.[20]). Similarly, there is no support for decoupling material from geometry definition and thus no ability to specify material templates that can be reused (e.g., repeating a pattern that defines a composite material, or defining a procedural gradation for functionally graded materials).

We think the right way to drive multimaterial 3D printers is a programmable synthesis pipeline, akin to the rendering pipeline. Instead of a static mesh per piece of material, OpenFab describes a procedural method to synthesize the final voxels of material at full printer resolution on demand. This provides efficient storage and communication, as well as resolution independence for different hardware and output contexts. It also decouples material definition from geometry. A domain-specific language and pipeline features specific to 3D printing make it much easier for users to specify many types of procedurally printed output than they could by writing standalone programs for every different material or fabrication application.

The OpenFab pipeline offers an expressive programming model for procedurally specifying the geometry and material of printable objects. A scene graph describes geometry and attributes, although *fablets* procedurally modify the geometry and define the material composition much like shaders in the rendering pipeline. Fablets are written in a domain-specific language (OpenFL) and provide a flexible toolset that supports many common material specification tasks.

We also propose a scalable architecture for implementing the OpenFab pipeline. As the total computational cost is large and it is impossible to fit the entire output volume into memory, the pipeline is designed to progressively stream output to the printer with a minimal up-front precomputation and with only a small slab of the volume kept in memory at any one time. An OpenFL compiler analyzes and transforms the procedural computation described by the fablets as needed for efficient implementation in the fabrication pipeline.

We evaluate the system on a variety of multimaterial 3D objects that have been specified and computed using our pipeline. We discuss how our system can be used to easily describe metamaterials, graded materials, and objects that contain materials with varied appearance and deformation properties. We print a number of results using a commercial multimaterial 3D printer and evaluate the performance of our prototype implementation.

2. BACKGROUND AND RELATED WORK

"3D printing" is an umbrella term for a variety of additive manufacturing processes where parts are built up from constituent materials, typically one layer at a time, in an incremental fashion. Processes vary by what kind of materials

The original version of this paper appeared in *ACM Transactions on Graphics 32*, 4 (July 2013).

they work with (polymers, ceramics, metals, etc.) and how the material itself is deposited and bonded together. Some processes such as PolyJet, stereolithography (SLA), selective laser sintering (SLS), and powder-binder printing have direct control over the geometry of the part at the individual voxel level. Other processes such as fused deposition modeling (FDM)—which has been popularized by entry-level desktop systems from MakerBot, Ultimaker, and others—perform material deposition through a continuous vector motion that does not allow for precise voxel-level control. Further, some 3D printing processes allow for the precise deposition of *different* materials at the voxel level, most notably the Connex systems from Objet and the 3D Systems ProJet MJP 5600.

Our work specifically targets state-of-the art 3D printing processes that allow for precise deposition of different materials at each voxel. Such processes can print at resolutions higher than 300 dots per inch, where each printed layer can be represented as a bitmap, and each pixel in the bitmap specifies the particular material type that needs to be deposited at that location. All layers together define on the order of 10^8 voxels per cubic inch.

Traditionally, 3D printing has synthesized uniform material objects defined by unstructured surface meshes.[1] Multiple materials are supported by statically assigning a single material to each mesh. Various companies have created proprietary formats to support their specific equipment. Nevertheless, with current printing software, it is unclear how the geometric data is translated to machine instructions, making the printing process difficult to control from outside. Open-source efforts (e.g., RepRap and Fab@Home) largely target FDM printing processes, which are motion vector-based, low-resolution, and low-throughput architectures, with limited support for multiple materials (multiple materials are handled as separate STL files). The Additive Manufacturing File Format (AMF) standard[3] allows description of object geometry, its composition, and color. Colors and materials can be specified with limited proceduralism, using simple expressions from voxel coordinates to material choices, but these have limited expressive power, and no architecture has been proposed to efficiently implement this model.

In contrast to the model-oriented descriptions supported by traditional 3D printing software, standard APIs and formats in 3D rendering and 2D printing describe how an output device should synthesize an image.[2, 5, 11, 16, 17] Rendering pipelines, in particular, balance flexibility and efficient implementation by combining a fixed pipeline with user-programmable stages, and programmable shaders decouple geometry from material description. Our fabrication pipeline is inspired by the success of programmable rendering pipelines and uses shader-like *fablets* to describe microgeometry and material composition. Our scene description parallels standard scene graph representations,[4] with extensions specific to fabrication, and without many complexities necessary for animation and interactivity. More detailed treatment of related work is given in the original version of this paper.[19]

3. DESIGN PHILOSOPHY

Mixing many materials with different optical and mechanical characteristics at inkjet printer resolution allows extremely complex objects with countless unique and spatially varying properties to be synthesized directly from a digital description. At the same time, print volumes and speed are growing, although cost is falling, putting additive multimaterial manufacturing within reach of much more applications. These trends led us to several major principles, which guided our design as follows:

- **Continuous material definition.** To unlock the full capabilities of printer hardware, our system should allow continuous material definition at full printer resolution.
- **Streaming architecture.** In order to achieve scalability necessary for printing large build volumes at native resolution, the OpenFab pipeline should only use local storage and computation wherever possible, streaming over the output volume in the order required by the printer. It should also require as little up-front precomputation as possible, to minimize printer startup delay.
- **Procedural synthesis.** Expressive tools, especially a shader-like language and programming model, provide a more natural way to describe complex optical and mechanical material logic than is currently possible with static meshes per material. Procedural synthesis also supports scalability, trading memory for computation: the material composition and geometric detail do not have to be stored explicitly but can be computed procedurally, as required by the printer.
- **Decoupling material from geometry.** Complex material logic should be defined independently of the mesh geometry and be reusable across models.
- **Automatic adaptation to hardware.** Procedural synthesis of surface and volume detail provides resolution independence for different output sizes and resolution. Automatically normalizing and dithering the multimaterial mixtures, accounting for physical constraints such as different materials expanding or contracting when cured, dramatically simplify the development of device-independent procedural materials.

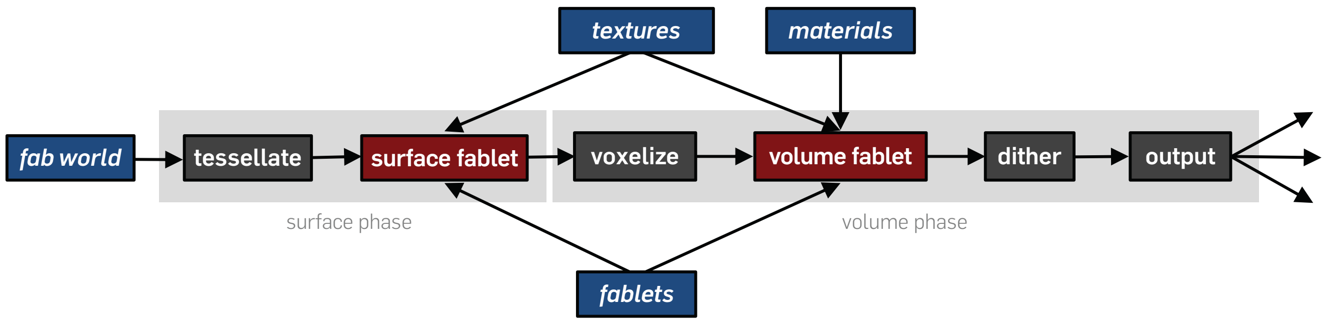
4. PROGRAMMING MODEL

To meet these design goals, we propose a programmable pipeline abstraction for 3D printing (Figure 1). Similar to rendering pipelines, some of the stages are fixed and others are programmable by the user. The role of the pipeline is to process a combination of geometric input, image textures, and *fablets* to synthesize device-specific fabrication output. The user controls the process by defining geometry and textures, setting pipeline attributes and options, and writing fablets. User-programmable fablets procedurally transform and compute the attributes at each vertex of the object mesh and compute the material mixture output at each point within the mesh volume.

4.1. Pipeline input

The input to the pipeline is a *fab world*, a scene graph-like description that consists of object boundary representations and associated attributes such as transforms, image texture inputs, and fablets. Recurring shapes can be defined

Figure 1. The OpenFab pipeline defines a programming model for synthesizing continuous volumetric material mixtures for 3D printing. As an input (blue), it takes a scene graph describing a set of object boundary representations, textures, printer materials, and user-programmable *fablets*—similar to shaders. From this input, it generates a discrete volumetric material definition that is device specific. Some stages are fixed function (gray), controlled by high-level parameters and printer characteristics, although *fablet* stages (red) are programmable by the user.



and *instanced* multiple times. Objects are also given *priorities*, which allow CSG-like operations and control the treatment of overlapping regions. (If two or more objects cover the same voxel, that voxel is assigned to the object with a higher priority value.)

4.2. Surface phase

In the first stage of the pipeline, the surface of the input objects is discretized via *tessellation* to match the desired output resolution. Tessellation generates micropolygon primitives—the common surface representation throughout the pipeline—with user-defined attributes interpolated from the input object.

Next, the *surface stage* of the user's fablet program is evaluated over these micropolygons. The surface fablet stage is conceptually similar to a vertex shader: it evaluates point-wise, given a vertex position and interpolated attributes as input, and performs arbitrary computation, such as sampling from textures, to produce a (potentially displaced) output position and some number of output attributes. Procedural displacement provides a mechanism for describing surface microgeometry in arbitrary detail that would be infeasible to explicitly model in the input objects.

4.3. Volume phase

The next stage discretizes the volume enclosed by objects via *voxelization*. It is critical to use consistent, crack-free rules for voxelization. We follow the rules of 26-separating voxelization.[7] Consider a multipart assembly where parts are printed separately. To ensure the assembly fits together, each voxel overlapping the shared geometry along part boundaries must be consistently assigned to exactly one part.

After voxelization, the *volume stage* of the fablet is evaluated over each voxel. This stage receives interpolated attributes as input and performs arbitrary computation to produce a floating point-valued mixture of the available printing materials. In addition to standard arithmetic, control flow, and texture sampling functionality, the volume fablet stage also allows querying the distance to the nearest point on the surface, and any surface attributes at that point. When defining materials volumetrically, it is often useful to be able to determine the relative position of a given voxel with respect to the object boundary.[12] Consider the scenario where we would like to print a textured object. Unlike rendering, we cannot assign colors simply to the infinitesimal outermost layer of the surface; rather, the printer needs to deposit a certain volume of layered material in order to achieve a particular color, reflectance, and scattering behavior.

Finally, the *dither* stage performs volumetric quantization and discretization of material quantities to match the capability of the target printer before handing the result to a specific printer driver. Our initial implementation included a streaming raster slice backend that is appropriate for a drop-on-demand 3D printers and a legacy backend that generates per-material STL meshes for use with traditional commercial printer software.

4.4. Fablets and OpenFL

Fablets are written in OpenFL, a C-like programming language similar to shader languages such as HLSL.[5] Unlike most shader languages, OpenFL describes both surface and volume functionality together as methods on a single `fablet` object. Uniform parameters, such as texture and material IDs, are also declared in the object. OpenFL includes standard functions for common math, texturing, and other routines. Unique to our domain, it also includes functions to query the distance to the nearest point on the surface, as well as any interpolated mesh attributes at that point.

To understand how fablets can be used to define procedural surface detail and continuous volumetric material variation, consider the example as shown in Figure 2. One side is flat and texture mapped with the foreground image, whereas the other side is displaced according to the desired brightness of the shadow background image. This object is defined by the following fablet:

Material and texture handles are declared as attributes of the fablet, along with parameters for the dimensions of the rectangular border, maximum thickness, and the depth into the volume to which the texture should be deposited on the front face.

The surface phase takes as arguments the position, normal, and texture coordinates defined over the mesh, as well as a per-vertex flag indicating the face of the cube (front, back, or side). If the currently processed vertex is on the back

```
fablet MagicPostcard {
  @uniform {
    float2 border;
    float textureDepth, maxThickness;
    ImageTexture2D fg, bg;
    Material white, black;
  }

  const int CARD_FRONT = 0, CARD_BACK = 1;

  @Surface(@varying {
              SurfaceAttributes attr,
              float2 uv, int face,
              out float2 uvOut, out int faceOut
            })
  {
    // pass through attributes
    uvOut = uv;
    faceOut = face;
    if (face == CARD_BACK) { // backface
      float L = bg.Sample1(uv[0], uv[1], 0);
      float thickness;
      if (uv[0] < border[0] || uv[0] > 1 - border[0] ||
          uv[1] < border[1] || uv[1] > 1 - border[1]) {
        thickness = maxThickness;
      } else {
        // material approximation: transmission
        // has quadratic falloff with thickness
        thickness = sqrt(1 - L) * maxThickness;
      }
      return attr.n * thickness;
    } else {
      // no displacement on the front and sides
      return 0;
    }
  }

  @Volume(@varying {
              VolumeAttributes attr,
              @nearest float2 uv,
              @nearest int face
            })
  {
    MaterialComposition mc;
  if (face == CARD_FRONT && // front face
      abs(distance(attr.voxelCenter)) <= textureDepth) {
    // surface texture
    float L = fg.Sample1(uv[0], uv[1], 0);
    mc.Set(white, L);
    mc.Set(black, 1 - L);
  } else {
    // background/border
    mc.Set(white,1);
  }
  return mc;
 }
}
```

face, the fablet computes a material thickness based on the luminance of the background image and displaces the mesh accordingly. It creates a fixed-depth border in a narrow band around the edges defined by the `border` parameter. Outside the back face, it performs no displacement and simply returns the original vertex position.

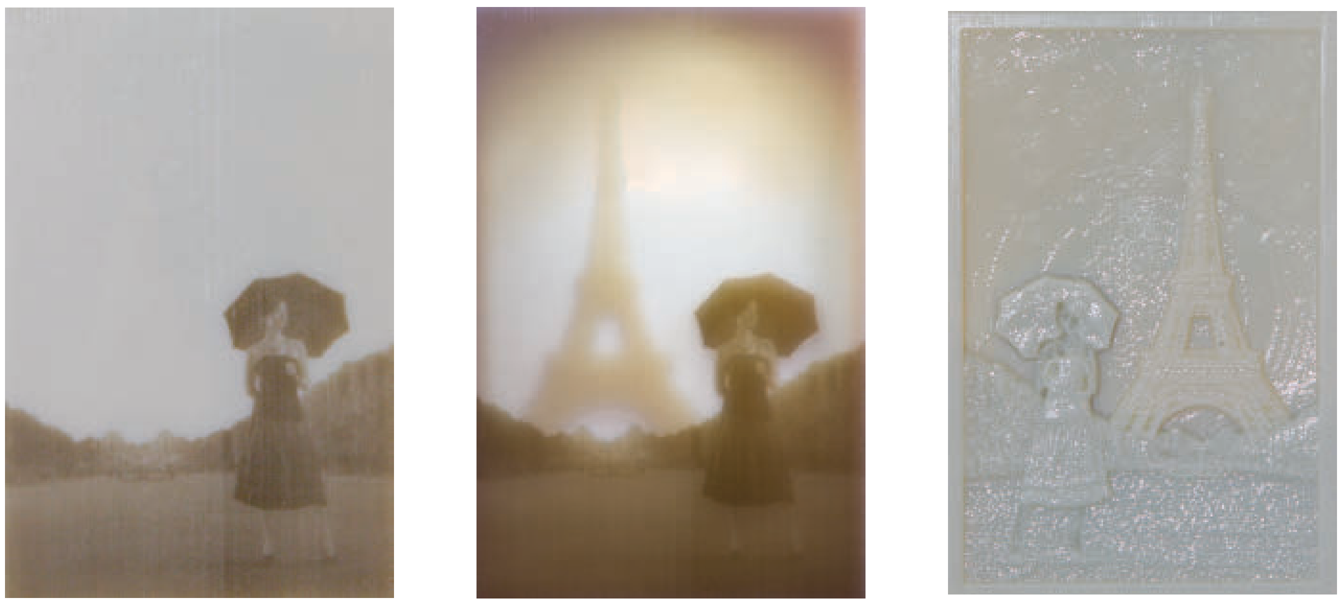
The volume phase takes as its argument the 3D position of the center of the currently processed voxel. It then uses the `face` flag from the *nearest* surface point to determine if it is near the front face. If it is and the `distance` to the surface is within `textureDepth`, it samples the foreground image texture based on the nearest surface texture coordinates and mixes black and white materials based on the brightness at that point. Note that the texture cannot simply be deposited in an infinitesimal layer on the surface. To show up clearly in real materials, it is usually necessary to deposit colors down from the surface to some depth inside the interior volume. Elsewhere in the object, it outputs plain white material.

5. ARCHITECTURE

The OpenFab pipeline is inspired by Reyes[8] and modern real-time rendering APIs such as OpenGL and Direct3D, and it is similarly designed to facilitate efficient implementation. Specifically, it is designed to allow a streaming implementation, starting to produce output quickly after startup and driving the printer on demand within a fixed and controllable memory footprint. Additionally, the fablet programming model is designed to admit data parallel computation in the same style as shaders in rendering.

Our reference implementation was built to stream output with a fixed memory budget and low startup time. It is a scalable foundation for a high performance implementation, but many individual stages are internally unoptimized.

Figure 2. The front face of the postcard (left) is texture mapped using a foreground image. The back (right) displaces the surface to create a spatially varying transmission according to a combined foreground and background image. The result is a hidden background image, which only appears when backlit (center).



Nonetheless, it is more than fast enough to keep up with currently available printers.

The architecture of our implementation is shown in Figure 3 and proceeds as follows:

```
Precompute acceleration structures
for each slab, in printer order:
    for each shape overlapping slab:
        Compute surface microgeometry and attributes
        Compute voxels and material composition
    Normalize and dither materials to device capability
        Output slab to printer
```

5.1. Setup

We begin by calculating bounds for each shape in the scene. Because fablets can displace surface geometry, this is not known directly from the input. Users provide maximum displacement bounds, but we additionally execute an interval arithmetic variant of the surface fablet stage to automatically infer displacement bounds as well[6] and use the minimum of the user-provided and inferred bound.

We next create acceleration structures to speed up the nearest surface point queries performed in the volume fablet stage. We build a bounding volume hierarchy (BVH) that spatially partitions the base primitives of the input mesh, conservatively accounting for possible displacement using the displacement bounds calculated in the prior stage. This upfront process is fast because it is performed on the untessellated base primitives of the input.

If the target printer requires support structures, we precalculate the places where such support is needed. We use a fast, high-resolution, fixed-point rasterizer to perform an orthographic Z-buffer rendering along the print platform movement (z) axis. We conservatively dilate each primitive to account for any possible displacement using the bounds calculated in the first stage. The resulting depth map contains the highest point along the z axis at which the material is present for each voxel column represented by the given depth sample. During the later output phase, if a given voxel is void we output support if and only if the height of that voxel is lower than the highest populated voxel for that particular voxel column as recorded in the depth map (Figure 4).

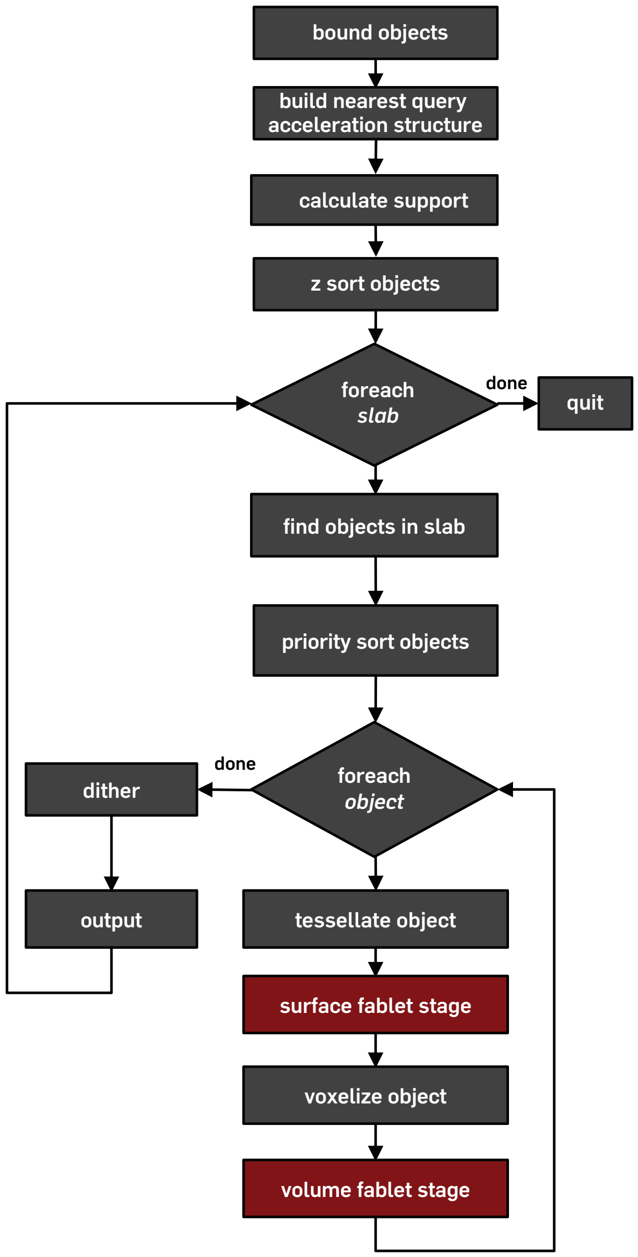
5.2. Slab processing *(outer loop)*

To begin printing, we subdivide the print volume into slabs. The size of the slab is dynamically calculated based on target memory usage and is a function of the resolution of the print and the total build volume. As we process each slab, we maintain a working set of shapes whose bounding volume intersects the current slab. As we begin the processing of each slab, we update the working set by removing shapes that are now beyond the current slab and adding ones that are now under the slab's domain.

5.3. Shape processing *(inner loop)*

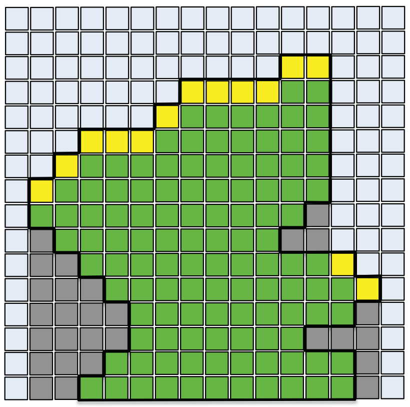
Within the working set, we sort objects by user-provided priority, processing from the highest to lowest priority and immediately discarding any newly generated voxels that are already occupied. Early culling makes fablet evaluation efficient: only one fablet (the one assigned to the highest priority object) gets evaluated per voxel.

Figure 3. The architecture of the OpenFab implementation is designed to stream over large, high-resolution print volumes with a fixed memory budget. The printing volume is divided into *slabs* along the primary printer axis, sized to bound memory usage. The pipeline processes one slab at a time and streams the output to the printer. Minimizing the amount of precomputation before streaming begins keeps startup time to a minimum, letting the printer start working almost immediately after OpenFab begins processing. Intermediate results such as tessellated geometry that span slab boundaries are cached for reuse, and the caches are also set to a fixed maximum size.



The first stage of the per-shape loop performs partial tessellation. Primitives can also be tessellated on demand in order to perform the distance function or the nearest surface attribute queries. We always tessellate into micropolygons, our common 2D primitive for the remainder of the

Figure 4. A 2D representation of our support generation approach. Voxels in green and yellow are part of the object being printed. Voxels in gray are support voxels. Voxels in yellow are part of the depth map that is generated with a high-resolution, fixed-point rasterizer. Support voxels are generated for empty voxels if there is a voxel in the depth map above them.



pipeline. Tessellated primitives are cached and reused if the primitive straddles multiple slabs or is accessed by multiple fablet queries. The cache has a fixed memory budget, managed with a simple LRU policy.

We next evaluate the surface phase of the fablet on the resulting tessellated mesh. We evaluate a quad at a time in order to compute derivatives (e.g., for texture filtering) and use OpenImageIO as our texture engine.[10]

Given a processed surface, we perform solid voxelization of its intersection with the current slab at the desired output resolution. We evaluate the volume phase of the fablet for each covered voxel. The underlying voxel grid is optimized to store up to 16 materials (out of a total of 64 materials that can be defined in the fab world) to allow each voxel to be compactly encoded in just 4 bits.

Surface distance and attribute queries are evaluated on demand by searching the corresponding acceleration structure. To allow fast startup, the acceleration structure encodes base mesh primitives (expanded conservatively to account for displacement bounds). At search time, candidate base primitives are tessellated and displaced by the surface fablet, and their microgeometry recursively searched for the nearest point or attributes. The results of tessellation and fablet evaluation are cached in the posttessellation surface cache, so that they are rarely recomputed. The cache size trades off memory overhead with redundant recomputation of surface geometry accessed in multiple places.

5.4. Output

When mixing multiple materials, we discretize the final output to a single material per voxel by applying Floyd-Steinberg dithering[9] to each slice at the same resolution as the voxelized grid. Error diffusion achieves a natural balance: if the fablet outputs one material, the dithered output matches the resolution of the printer, whereas if the fablet outputs multiple materials, the dithered output gracefully reduces the resolution in order to achieve the requested material ratios. A sliding window implementation reduces storage pressure for large slabs and satisfies our fixed memory requirements.

Finally, we output a custom raster format. Given the presence of multiple coordinate systems and resolutions within a given printer (e.g., from the motion system, linear encoders, arrays of printhead nozzles, variably sized droplets, different material properties), our native output is abstract enough that it allows a printer-specific backend to perform the necessary mapping to low-level commands that take these various sources of resolution into account. Alternatively, when targeting commercial printers that only take STL as input, we generate a set of boundary meshes for each material used, using a method similar to marching cubes.[13]

6. RESULTS

We have designed and fabricated a variety of different objects that highlight features of the OpenFab pipeline.

Our results were printed on an Objet Connex 500, a high-end multimaterial 3D printer that uses photopolymer phase-change inkjet technology and is capable of simultaneously printing with two primary materials and one support material. It supports a variety of polymer-based materials that vary in color, elasticity, and optical qualities. It takes per-material geometry meshes as an input. The build volume of the results is limited by the maximum number of primitives allowed by the Objet driver software—at most about 10 million.

Our first result, as shown in Figure 5, highlights the ability to easily apply different fablets to the same base geometry. The appearance of the rhinos varies significantly, and each uses a variety of features in OpenFab. For instance, the left rhino uses displacement mapping in the surface phase of the fablet to create microspikes over the rhino's skin. The volume phase of the fablet samples from a zebra-like texture to apply a layer of textured material near the surface. It uses the ability to query the nearest point to both retrieve the texture coordinate necessary to sample the texture and determine whether to apply the textured material. The center rhino has holes carved out throughout its body by returning void in the volume phase of the fablet. We use a distance function to separate the transparent outer shell of the rhino from the black inner core. The right rhino achieves its look in a similar fashion.

Our next result, the butterfly (Figure 6), highlights the use of object priority to achieve a CSG difference-like operation. The butterfly is placed within a transparent casing to simulate an amber fossil (the butterfly geometry has higher priority than the casing). We procedurally define volumetric cloudiness and particles in order to increase the appearance realism of the amber.

The bunny and the teddy bear pair (Figure 7) demonstrate the ability to reuse the same fablet across different models. The material used to print these objects is flexible but volume preserving. The fablet introduces procedurally defined and repeated void spaces in order to achieve a compressible, foam-like material. This demonstrates the ability to easily define and apply patterned materials. One could also make the 2D or 3D pattern be texture-driven. OpenFab allows one to build a library of such fablets similar to how material and light libraries are built for image rendering.

The magic postcard (Figure 2) demonstrates a creative use of texture-driven displacement mapping in its fablet (code

Figure 5. Three rhinos defined and printed using OpenFab. For each print, the same geometry was paired with a different *fablet*—a shader-like program that procedurally defines surface detail and material composition throughout the object volume. This produces three unique prints by using displacements, texture mapping, and continuous volumetric material variation as a function of distance from the surface.

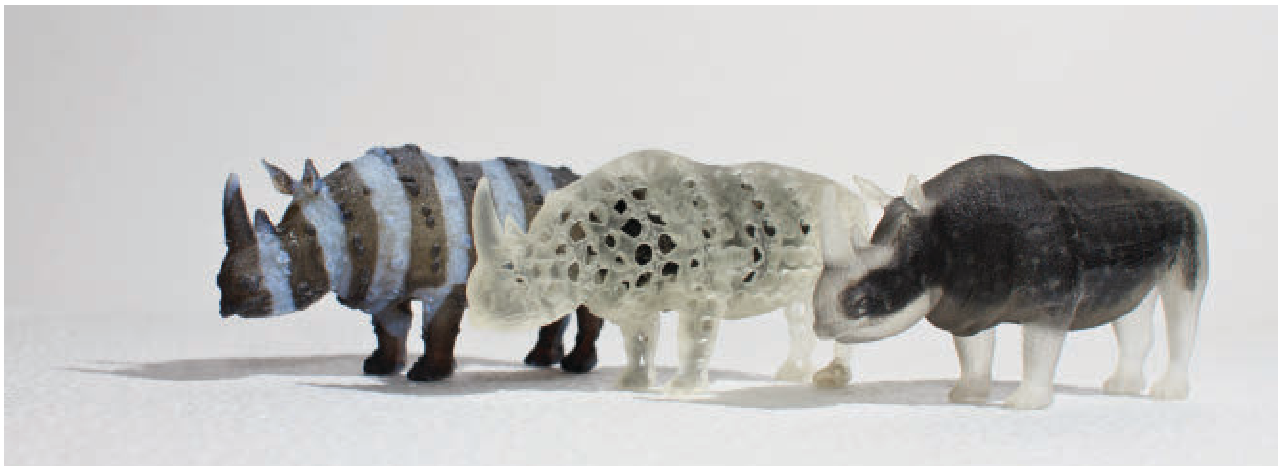
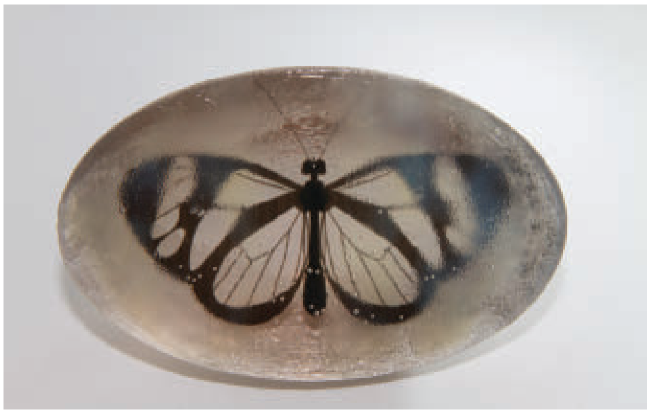


Figure 6. Insect embedded in amber. Object priority is used to embed the procedurally displaced insect mesh inside the outer amber hemisphere. The amber region mixes small amounts of white material according to procedural noise to model cloudiness and variation in the amber.



in Section 4.4). The front face of the postcard (shown left) is textured using a foreground layer of image texture. The back of the postcard (shown right) displaces the surface to create a spatially varying transmission effect. The amplitude of the displacement at each point is driven by the luminance of the background image. When illuminated solely from the front, the background layer is not visible. When another illumination source is added from the back, the whole image becomes visible (shown center). Similar to other textured objects, the postcard fablet uses the nearest point query and distance from the surface to perform texture-driven material assignment.

The marble table in Figure 8 (center) procedurally recreates the appearance of marble. It uses Perlin noise[15] to define a solid texture in the volume phase of the fablet. Note that the material distribution changes continuously to create a graded material.

The microlens in Figure 8 (right) demonstrates a working, procedurally defined microlens array. The surface phase of the fablet transforms a slab of material into an array of aspherical lenses by using displacement mapping. The volume phase of the fablet adds baffles in between the lenslets and assigns the two materials used (clear for lenses and black for the baffles). The baffles reduce the light leakage between the neighboring lenses.

Finally, in Figure 8 (left), we show two examples of objects made of procedurally defined materials with anisotropic mechanical properties. The core of the material is made of transparent and elastic material. We procedurally insert helical (top) or straight (bottom) rods made of white and rigid material. These rods influence the mechanical behavior: the helical rods allow twisting motion of the object in clockwise direction and very little twist in the opposite direction; the straight rods transform downward side pressure into transverse motion that causes elongation.

6.1. Performance

We ran a number of simulations to test the scalability of our initial implementation and its ability to provide fabrication data in real time to a 3D printer. Even without significant optimization, the initial OpenFab implementation meets our key design goals:

- Across many sizes of slices, it can stream the data as fast or faster than a high-end, multimaterial 3D printer can output material (in our case, an Objet Connex 500).
- Time to first slice output is seconds, even for models large enough to require hours to print.
- Memory footprint can be kept under a modest configurable threshold of 1.5GB without sacrificing these performance requirements.

Still, we observe that a significant amount of our runtime is spent in the nearest distance and nearest point queries and believe that a combination of optimized implementation of these and other key operations with data-parallel code generation could easily provide at least an order of magnitude performance increase to keep up with foreseeable future printers.

Figure 7. A procedurally defined foam material makes the bunny and bear squishy. Color and squishiness vary procedurally over the models.

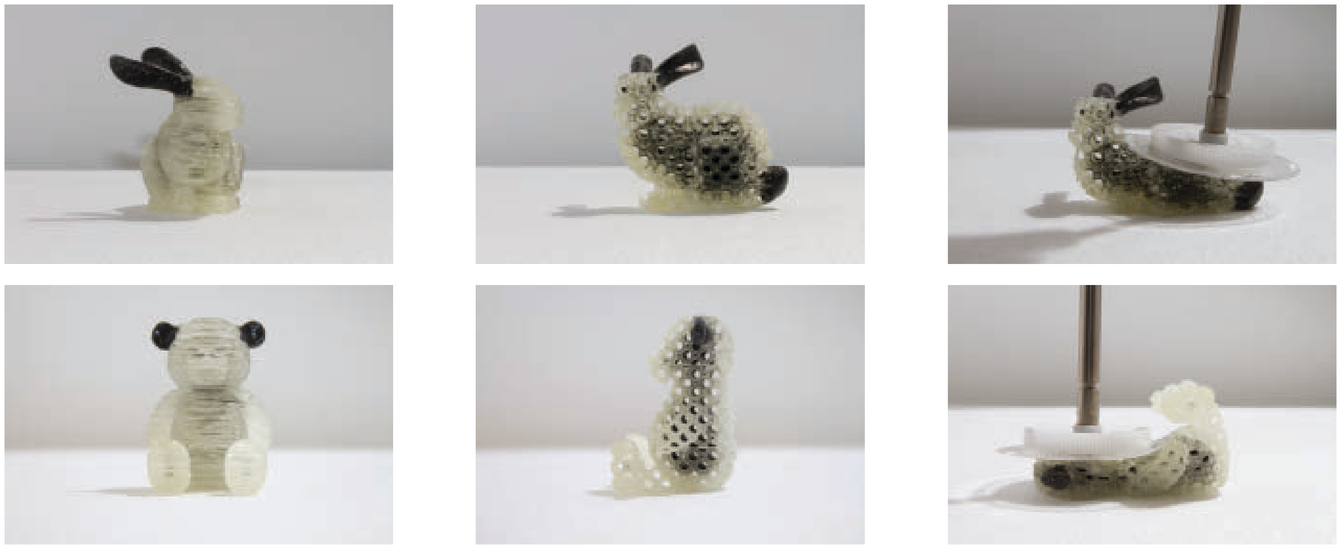
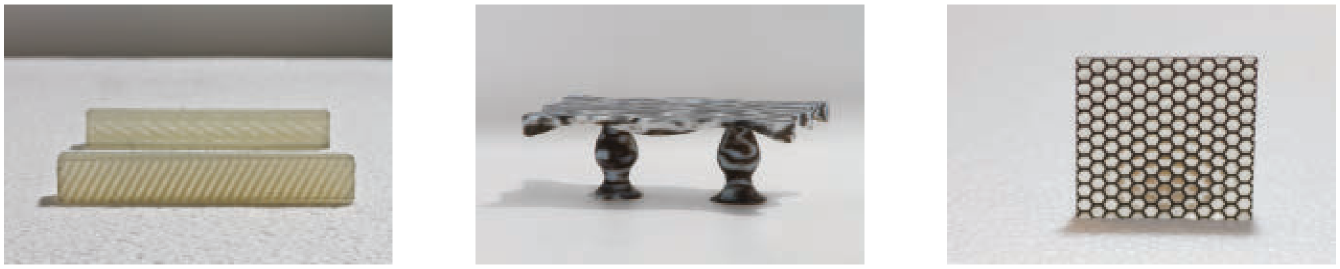


Figure 8. *Left*: procedurally defined materials with anisotropic mechanical properties. *Center*: Marble-like material generated using Perlin noise. *Right*: Procedurally defined and fully parameterized aspherical microlens array with baffles.



7. DISCUSSION AND FUTURE WORK

We have found the programmable pipeline abstraction a surprisingly powerful way to describe complex multimaterial 3D prints with a wide range of mechanical and optical properties. We think the OpenFab pipeline provides a solid and scalable foundation on which to build many multimaterial fabrication techniques.

The current programming model is powerful, but it is not the most natural way to describe all possible results. In the future, we think there is a great opportunity to spread proceduralism throughout the pipeline. Procedural geometry plugins could be more natural than the existing fablets for some types of geometry (e.g., synthesizing L systems) and would be complementary to the existing stages. Programmable dithering could also increase the flexibility of the pipeline and the degree of user control over the exact printed output.

Designing a full ecosystem around this pipeline is a natural direction for follow-up work. This could include a procedural modeling tool, a visual fablet authoring tool, and a print preview based on measured material properties. It is also desirable to extend the pipeline to integrate various mesh optimizations for automatic partitioning of large prints[14] and automatic detection and correction of structural stability.[18]

Performance is another area of possible future work. Our current implementation is more than fast enough to keep up with current printers. But, as printers get faster, build volumes grow, and fablets become more complex, it will be important to improve performance. Fortunately, there is an enormous room for optimization and parallelization in our implementation. The nearest surface queries from the volume fablet phase are a major component of our programming model and the single most expensive operation in our implementation. There is an opportunity to make these queries more efficient. Further, it will be interesting to define more complex surface-volume attribute relationships, such as alternative attribute interpolation methods.

Finally, native backends for many types of printer hardware will be important to realizing the full potential of the OpenFab pipeline. OpenFab was designed from the outset to drive continuous material output at full printer resolution. Current commercial printer software, however, is limited to STL format input and fails when given more than a few million polygons. This significantly limits the scale of spatially varying output we can feed to current commercially available printers. The printer backends, however, take raw full-resolution bitmaps of each slice. Interacting with printers at the raster level will allow streaming prints of continuous material variation at much larger scale.

Given the high-frequency details in dithered multimaterial slices, implementing a backend for vector path 3D printers (e.g., FDM) remains a challenge. Recent work on "multiplexer" extruders that combine multiple filaments is promising, though. We imagine targeting such printers by using dither masks that map local dither patterns to linearly weighted combinations of the input filaments.

Acknowledgments

We would like to thank Jaakko Lehtinen and Mark Leone for providing seed code for our fixed-point rasterizer and OpenFL compiler, respectively; Mark Leone and Frédo Durand for extensive feedback on writing; Pitchaya Sitthi-Amorn for providing feedback during design discussions; and Ye Wang and Moira Forberg for helping produce our results. Steve Marschner provided helpful guidance during revision for *Communications*. Some models are from TurboSquid, and the Eiffel Tower photo was obtained through 123RF.

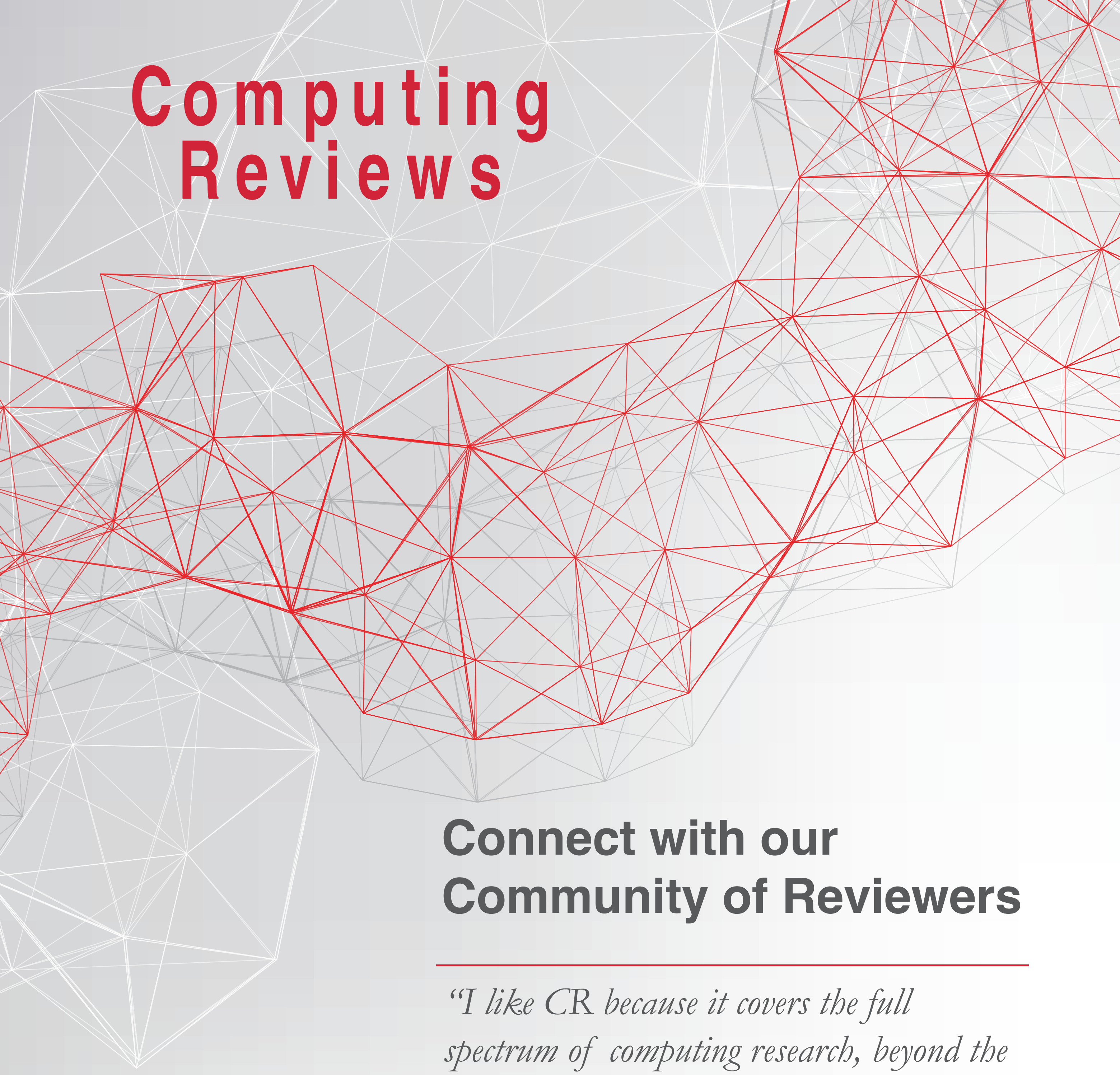
This work was funded in part by NSF grants CCF-1138967 and IIS-1116296, and DARPA grant #N66001-12-1-4242. K. Vidimče was supported by an NSF Graduate Research Fellowship and S. Wang by the MIT Undergraduate Research Opportunities Program.

References

1. 3DSystems. StereoLithography interface specification, 1988.
2. Adobe Systems. PostScript language reference, 1985.
3. ASTMStandard. Standard specification for additive manufacturing file format (AMF) version 1.1., July 2011.
4. Bell, G., Parisi, A., Pesce, M. *The Virtual Reality Modeling Language Version 1.0 Specification*. Technical Report, 1995.
5. Blythe, D. The Direct3D 10 system. *ACM Trans. Graph. 25*, 3 (July 2006), 724–734.
6. Clarberg, P., Toth, R., Hasselgren, J., Akenine-Möller, T. An optimizing compiler for automatic shader bounding. *Computer Graphics Forum 29*, 4 (2010), 1259–1268.
7. Cohen-Or, D., Kaufman, A. Fundamentals of surface voxelization. *Graph. Models Image Process. 57*, 6 (1995), 453–461.
8. Cook, R.L., Carpenter, L., Catmull, E. The Reyes image rendering architecture. In *Proceedings of SIGGRAPH*, ACM, New York, NY, USA, 1987, 95–102.
9. Floyd, R. Steinberg, L. An adaptive algorithm for spatial gray scale. In *Proceedings of the Society of Information Display. 17*, 2 (1976), 75–77.
10. Gritz, L. OpenImageIO 1.0. http://openimageio.org, 2012.
11. Hewlett-Packard. Printer command language, 1984.
12. Liu, H., Maekawa, T., Patrikalakis, N., Sachs, E., Cho, W. Methods for feature-based design of heterogeneous solids. *Computer-Aided Design 36*, 12 (2004), 1141–1159.
13. Lorensen, W. E., Cline, H. E. Marching cubes: A high resolution 3D surface construction algorithm. In *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques*. ACM, New York, NY, USA, 1987, 163–169.
14. Luo, L., Baran, I., Rusinkiewicz, S., Matusik, W. Chopper: Partitioning models into 3D-printable parts. *ACM Trans. Graph. 31*, 6 (Nov. 2012), 129:1–129:9.
15. Perlin, K. An image synthesizer. In *Proceedings of SIGGRAPH*. ACM, New York, NY, USA, 1985, 287–296.
16. Pixar. The RenderMan Interface. Technical report, 11, 2005.
17. Segal, M., Akeley, K. The OpenGL Graphics System: A Specification, Version 4.3. Technical Report, SGI, 2012.
18. Stava, O., Vanek, J., Benes, B., Carr, N., Měch, R. Stress relief: Improving structural strength of 3D printable objects. *ACM Trans. Graph. 31*, 4 (July 2012), 48:1–48:11.
19. Vidimče, K., Wang, S.-P., Ragan-Kelley, J., Matusik, W. OpenFab: A programmable pipeline for multi-material fabrication. *ACM Trans. Graph. 32*, 4 (2013).
20. Wang, L., Lau, J., Thomas, E.L., Boyce, M.C. Co-continuous composite materials for stiffness, strength, and energy dissipation. *Adv. Mater. 23*, 13 (2011), 1524–1529.

Kiril Vidimče, Szu-Po Wang, and Wojciech Matusik, Massachusetts Institute of Technology, Cambridge, MA, USA.

Jonathan Ragan-Kelley, University of California, Berkeley, USA.



California State University, Sacramento

Department of Computer Science

Tenure-Track Assistant Professor

One tenure-track assistant professor position to begin with the Fall 2020 semester.

Applicants specializing in any area of computer science will be considered. Those with a strong background in computer science or computer engineering subject areas are especially encouraged to apply. Ph.D. in Computer Science, Computer Engineering, or closely related field required by the time of the appointment.

For detailed position information, including application procedure, please see https://csus.peopleadmin.com/. Screening will begin September 16, 2019 and remain open until filled.

AA/EEO employer. Clery Act statistics available. Mandated reporter requirements. Criminal background check will be required.

Furman University

Open-Rank Tenure Track Professor in Computer Science

The Department of Computer Science at Furman University invites applications for a tenure track position at the Assistant, Associate, or Full rank to begin August 1, 2020. Candidates must have a Ph.D. in Computer Science or a closely related field, and all areas of specialty will be considered. The position requires teaching excellence, scholarly and professional activity involving undergraduates, effective institutional service, and a willingness to work with colleagues across disciplines.

The Department of Computer Science confers degrees with majors in Computer Science (B.S.) and Information Technology (B.S. and B.A.), an innovative, interdisciplinary program of study. The Department values teaching and research projects that bridge Computer Science with other disciplines, efforts to provide students with learning opportunities outside the classroom and in the community, and contributing to Furman's university-wide First Year Writing Seminar program. Furman Computer Science professors mentor undergraduates both formally and informally, and work to build a welcoming student-faculty community.

Furman University is a selective private liberal arts and sciences college committed to helping students develop intellectually, personally, and interpersonally and providing the practical skills necessary to succeed in a rapidly-changing world. Furman professors are exceptional teacher-scholars who mentor undergraduate students within a campus community that values and encourages diverse ideas and perspectives. Our recently-launched strategic vision, The Furman Advantage, promises students an individualized four-year pathway facilitated by team of mentors and infused with a rich and varied set of high-impact experiences outside the classroom that include undergraduate research, study away, internships, community-focused learning, and opportunities to engage across disciplines.

Furman is an Equal Opportunity Employer committed to increasing the diversity of its faculty and staff. The University aspires to create a community of people representing a multiplicity of identities including gender, race, religion, spiritual belief, sexual orientation, geographic origin, socioeconomic background, ideology, world view, and varied abilities. Domestic partners of employees are eligible for comprehensive benefits.

The Furman student experience is supported by a rich network of centers and institutes that includes The Riley Institute, The David E. Shi Center for Sustainability, The Institute for the Advancement of Community Health, The Rinker Center for Study Away and International Education, The Cothran Center for Vocational Exploration, The Shucker Center for Leadership Development, The Malone Center for Career Engagement, and our newest addition, The Center for Inclusive Communities.

Furman is located in Greenville, South Carolina, which is one of the fastest growing cities in the Southeast and is ranked among "America's Ten Best" by Forbes Magazine. The charming downtown features excellent restaurants, in-town parks, shops, museums, galleries, music venues, and theaters. The city also has excellent public and private schools and a vibrant international community. A 20-mile bike and running trail connects the university to Greenville and to Travelers Rest, which was named "one of America's coolest small towns." The surrounding area abounds with outdoor recreational activities and has some of the most beautiful lakes, rivers, and mountains in the country. Greenville is within easy reach of the Blue Ridge Mountains and Atlantic beaches. The newly renovated Greenville-Spartanburg Airport, located just 25 minutes from downtown, runs daily flights to major cities and airline hubs. Greenville is 2 1/2 hours from Atlanta and only one hour from Asheville, North Carolina. It is an ideal place to live and work.

Applicants should submit a curriculum vitae, cover letter, statement of teaching philosophy and experience, statement of research interests, an official copy of most recent transcripts, and a diversity statement that describes how your teaching, scholarship, mentoring and/or service might contribute to a liberal arts college community that includes a commitment to diversity as one of its core values. Three letters of recommendation should be sent separately. Review of applications will continue until the position is filled, but to ensure full consideration, applications should be completed by November 15, 2019. Questions can be directed to the chair of the

NUS National University of Singapore

Sung Kah Kay Assistant Professor in All Areas of Computer Science

The Department of Computer Science at the National University of Singapore (NUS) invites applications for the Sung Kah Kay Assistant Professorship. Applicants can be in any area of computer science. This prestigious chair position was set up by the family and friends of the late Assistant Professor Sung Kah Kay after his untimely demise early in his career at NUS. Candidates should be early in their academic careers and yet demonstrate outstanding research potential, and a strong commitment to teaching.

The Department enjoys ample research funding, moderate teaching loads, excellent facilities, and extensive international collaborations. We have a full range of faculty covering all major research areas in computer science and boasts a thriving PhD program that attracts the brightest students from the region and beyond. More information is available at *www.comp.nus.edu.sg/careers*.

NUS is an equal opportunity employer that offers highly competitive salaries, and is situated in Singapore, an English-speaking cosmopolitan city that is a melting pot of many cultures, both the east and the west. Singapore offers high-quality education and healthcare at all levels, as well as very low tax rates.

Application Details:

- Submit the following documents (in a single PDF) online via: *https://faces.comp.nus.edu.sg*
  - A cover letter that indicates the position applied for and the main research interests
  - Curriculum Vitae
  - A teaching statement
  - A research statement
- Provide the contact information of 3 referees when submitting your online application, or, arrange for at least 3 references to be sent directly to *csrec@comp.nus.edu.sg*

Application reviews will commence immediately. We hope to fill the position by January 2020.

If you have further enquiries, please contact the Search Committee Chair, Weng-Fai Wong, at *csrec@comp.nus.edu.sg*.

Department of Computer Science, Dr. Kevin Treu, at kevin.treu@furman.edu.

To submit an application and letters of recommendation, please visit https://furman.wd5.myworkdayjobs.com/en-US/Furman_Careers/job/Main-Campus/Assistant-Professor-Computer-Science-2_R000685.

Harvard John A. Paulson School of Engineering and Applied Sciences (SEAS)

Tenure Track Faculty Position in Computer Science

The Harvard John A. Paulson School of Engineering and Applied Sciences (SEAS) seeks applicants for a position at the tenure-track level in Computer Science, with an expected start date of July 1, 2020.

We are accepting applications in all areas of Computer Science. Machine learning, natural language processing, systems, systems security, and algorithms are areas of special interest, but candidates in any area are invited to apply.

We seek candidates who have a strong research record and a commitment to undergraduate and graduate teaching and training. We particularly encourage applications from historically underrepresented groups, including women and minorities.

Computer Science at Harvard benefits from outstanding undergraduate and graduate students, world-leading faculty, an excellent location, significant industrial collaboration, and substantial support from SEAS. Information about Harvard's current faculty, research, and educational programs in computer science is available at http://www.seas.harvard.edu/computer-science.

The associated Institute for Applied Computational Science (http://iacs.seas.harvard.edu) and Data Science Initiative (https://datascience.harvard.edu/) foster connections among computer science, applied math, data science, and various domain sciences at Harvard through its graduate programs and events.

A doctorate or terminal degree in Computer Science or a related field is required by the expected start date.

Required application documents include a cover letter, CV, a statement of research interests, a teaching statement, and up to three representative papers. In addition, we ask for a statement describing efforts to encourage diversity, inclusion, and belonging, including past, current, and anticipated future contributions in these areas.

Candidates are also required to submit the names and contact information for at least three and up to five references, and the application is complete only when three letters have been submitted. At least one letter must come from someone who has not served as the candidate's undergraduate, graduate, or postdoctoral advisor. We encourage candidates to apply by December 6, 2019, but will continue to review applications until the position is filled. Applicants can apply online at http://academicpositions.harvard.edu/postings/9134.

Harvard is an equal opportunity employer and all qualified applicants will receive consideration for employment without regard to race, color, religion, sex, national origin, disability status, protected veteran status, gender identity, sexual orientation, pregnancy and pregnancy-related conditions, or any other characteristic protected by law.

Indian Institute of Science

Full Time Faculty Positions

The Division of EECS (Electrical, Electronics, and Computer Sciences) at India's top-ranked university, Indian Institute of Science, at Bengaluru, is hiring. We are looking for exceptionally bright and motivated faculty candidates, with an established record of high quality research and a strong commitment to teaching, at the assistant professor and associate professor levels.

Departments: The EECS Division comprises four departments: CSA (Computer Science and Automation), ECE (Electrical Communication Engineering), EE (Electrical Engineering), and ESE (Electronic Systems Engineering). In the EECS Division, we strive towards fundamental advances in a broad range of core areas as well as interdepartmental research-driven innovation in thematic clusters. We seek faculty applicants in the core areas as well as in the thematic clusters.

Core Areas: The core areas we pursue include but are not limited to: Theoretical Computer Science, Programming Languages and Software Engineering, Computer Systems, Machine Learning, Con-

The Graz University of Technology, Faculty of Computer Science and Biomedical Engineering,

TU Graz
Graz University of Technology

invites interested scientists to apply for the following faculty positions:

- **Professor of Computational Medicine** (full tenured position)
- **Professor of Computational Behavioral and Social Sciences** (full tenured position)
- **Tenure Track Professor in Visual Computing** (for women only)
- **Tenure Track Professor in Information Security**
- **Tenure Track Professor in Intelligent Systems**
- **Tenure Track Professor in Formal Methods for Dependabiltiy**
- **Tenure Track Professor in Machine Learning**

The Faculty of Computer Science and Biomedical Engineering at Graz University of Technology is committed to excellence in research and teaching. The faculty's research activities cover a broad spectrum of topics that are reflected in its main research areas, "Biomedical Engineering," "Safety and Security," "Intelligent Systems," and "Visual Computing." We are proud of our outstanding applied and fundamental research expertise and promote interdisciplinary projects. Moreover, our faculty fosters a close dialogue with industry and encourages the establishment of spin-offs.

Graz University of Technology aims to increase the proportion of women and therefore qualified female applicants are explicitly encouraged to apply. Graz University of Technology actively promotes diversity and equal opportunities. People with disabilities who have the relevant qualifications are expressly invited to apply.

Application deadline: **26 November 2019**. For details, see
https://www.tugraz.at/go/professorships-vacancies
or contact us at applications.csbme@tugraz.at

上海科技大学
ShanghaiTech University

TENURE-TRACK AND TENURED POSITIONS

ShanghaiTech University invites highly qualified candidates to fill multiple tenure-track/tenured faculty positions as its core founding team in the School of Information Science and Technology (SIST). We seek candidates with exceptional academic records or demonstrated strong potentials in all cutting-edge research areas of information science and technology. They must be fluent in English. English-based overseas academic training or background is highly desired.

ShanghaiTech is founded as a world-class research university for training future generations of scientists, entrepreneurs, and technical leaders. Boasting a new modern campus in Zhangjiang Hightech Park of cosmopolitan Shanghai, ShanghaiTech shall trail-blaze a new education system in China. Besides establishing and maintaining a world-class research profile, faculty candidates are also expected to contribute substantially to both graduate and undergraduate educations.

Academic Disciplines: Candidates in all areas of information science and technology shall be considered. Our recruitment focus includes, but is not limited to: computer architecture, software engineering, database, computer security, VLSI, solid state and nano electronics, RF electronics, information and signal processing, networking, security, computational foundations, big data analytics, data mining, visualization, computer vision, bio-inspired computing systems, power electronics, power systems, machine and motor drive, power management IC as well as inter-disciplinary areas involving information science and technology.

Compensation and Benefits: Salary and startup funds are highly competitive, commensurate with experience and academic accomplishment. We also offer a comprehensive benefit package to employees and eligible dependents, including on-campus housing. All regular ShanghaiTech faculty members will join its new tenure-track system in accordance with international practice for progress evaluation and promotion.

Qualifications:
- Strong research productivity and demonstrated potentials;
- Ph.D. (Electrical Engineering, Computer Engineering, Computer Science, Artificial Intelligence, Financial Engineering, Signal Processing, Operation Research, Applied Math, Statistics or related field);
- A minimum relevant (including PhD) research experience of 4 years.

Applications: Submit (in English, PDF version) a cover letter, a 2-page research plan, a CV plus copies of 3 most significant publications, and names of three referees to: sist@shanghaitech.edu.cn. For more information, visit http://sist.shanghaitech.edu.cn/2017/0426/c2865a23763/page.htm

Deadline: The positions will be open until they are filled by appropriate candidates.

trol and Optimization, Signal Processing, Cyber-Physical Systems, Image Processing, Computer Vision, Next-Generation Communications and Networking, Mm-Wave and THz RF Systems, Photonics, Quantum Technologies, Power Systems, Power Electronics, High Voltage Engineering, Electromagnetics, Semiconductor Technologies, and Micro/nanoelectronics.

Thematic Clusters: We are actively striving to attain leadership in thematic clusters that include Artificial Intelligence, Brain and Computation, Autonomous Systems, Quantum Computing and Communications, Microelectronics and VLSI Design, Cyber Security, Speech and Language Processing, Storage Systems, Visual Analytics, and 5G Systems.

The Institute provides attractive start-up grants and travel grants. Competitive, endowed faculty chairs are available to provide for additional resources. Most of the students in the Institute receive Government fellowships. Being located in Bengaluru, there are numerous opportunities for industry collaboration and to be a part of the lively startup ecosystem.

To learn more about the Division of EECS, we urge you to look up: https://eecs.iisc.ac.in.

We welcome applications from Indian citizens as well as foreign nationals. We especially encourage women candidates to apply. We strongly welcome Indian candidates belonging to reserved categories.

For more information on the application process, please visit: https://iisc.ac.in/about/faculty-corner/how-to-apply/.

Macalester College

Tenure-Track Position in Computer Science

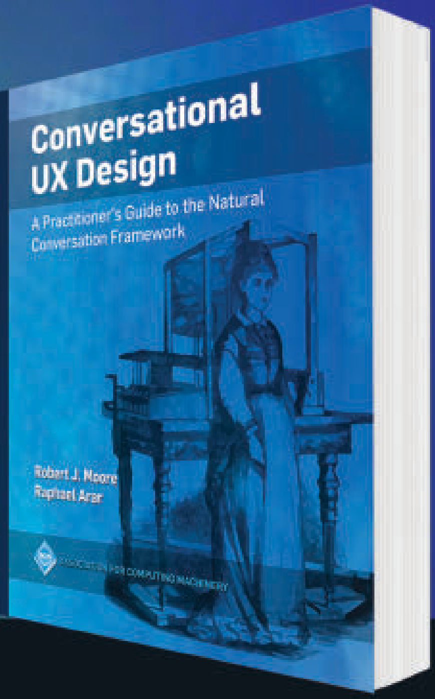
Macalester invites applications for a tenure-track position assistant professor level in Computer Science to begin Fall 2020. Candidates who have, or are completing, a Ph.D. in CS are preferred, but closely related fields may also be considered. Applicants must have a strong commitment to both teaching and research in an undergraduate liberal arts environment. We are especially interested in candidates who are enthusiastic to teach a broad range of undergraduate courses and mentor undergraduate research.

Areas of highest priority include mobile and ubiquitous computing, computer networks, algorithms and theory, programming languages, and data science. We encourage innovative pedagogy and curriculum and are interested in candidates whose work spans disciplinary boundaries.

Macalester's MSCS department (https://www.macalester.edu/mscs/) offers majors in Computer Science, Mathematics, and Applied Mathematics and Statistics, and minors in Computer Science, Mathematics, Statistics, and Data Science. MSCS is the largest department at the college with eleven Computer Science professors, seven of whom are tenure track. CS is the third largest major on campus, and over half of all students at the college take a CS course before they graduate. Class sizes range from 15 to 32 students and make extensive use of in-class teaching assistants. Most classes use active-learning formats centered around programming activities. In AY 2017 - 18, women accounted for 45% of graduating Computer Science majors. Seven of the eleven Computer Science faculty are women (and four of seven tenure track). On average 22% of both CS majors and, more broadly, students in CS courses, are US students of color. 30% of CS majors and enrollments are international students.

Macalester College is a highly selective, private liberal arts college in the vibrant Minneapolis-Saint Paul metropolitan area which has a population of roughly three million. It is nationally recognized as one of the best cities to live in the country, because of its thriving job market, beautiful park system, and dynamic art scene. Macalester's diverse student body comprises over 2000 undergraduates from 49 states and the District of Columbia and over 90 nations. The College maintains a longstanding commitment to academic excellence with a special emphasis on internationalism, multiculturalism, and service to society. As an Equal Opportunity employer supportive of affirmative efforts to achieve diversity among its faculty, Macalester College strongly encourages applications from women and members of underrepresented minority groups.

Applications require a cover letter, CV, transcripts, teaching and research statements, and three letters of recommendation. Details can be found at https://www.macalester.edu/mscs/compscitenure-trackjob/. Evaluation of applications will begin October 15th and continue until the position is filled.


Conversational UX Design
A Practitioner's Guide to the Natural Conversation Framework
Robert J. Moore
Raphael Arar
ASSOCIATION FOR COMPUTING MACHINERY

[CONTINUED FROM P. 112] that happens on the power lines. It turns out if you zoom into that noise source, it tells you a lot about what's happening. So our approach was to listen to all the electrical interference on the power line, then use machine learning to clarify and pattern-match to a specific device.

As it turns out, that's analogous to the water domain. When you flush the toilet or use the shower, you disrupt the water flow the moment you open and close that valve. And if you have a pressure sensor at any location, you can see a pressure wave that's indicative of the kind of valve that you just closed.

Given that most of us can't yet afford to live in "the home of the future" or put sensors onto all our fixtures and appliances, it's a refreshingly practical approach.

Sometimes, you have a scientific question where you are trying to address an algorithmic problem or find a more efficient way to do things, but at the same time you are also trying to think about how to apply it. If you come from a purely applied standpoint and you are solving an interesting problem, a lot of the scientific contributions follow, because you are now discovering new use-cases that you may not have discovered otherwise.

More recently, you have been working in healthcare, using commodity devices like mobile phones to do longitudinal and physiological monitoring.

We have looked at using the microphones to help people monitor respiratory ailments—so instead of using a dedicated device like a spirometer, say, you use machine learning and audio processing on the microphone to detect if something's happening in your respiratory system. We have also used the camera and flash to do non-invasive blood screening. You might take a picture of a baby to figure out how much bilirubin is in the blood and whether jaundice is a concern. You can't do a blood draw every single day, so having a non-invasive screening tool can be a really effective way to tell you when you should get to the next level of screening or diagnostics, and then you can intervene much sooner.

"If you come from a purely applied standpoint and you are solving an interesting problem, a lot of the scientific contributions follow, because you are now discovering new use-cases."

Yet capturing the right data to get ahead of a major health issue is incredibly difficult.

Most people see a doctor every one to two years. There can be a lot of indicators that could help you get ahead of a problem well before you are symptomatic. The challenge is, we don't have access to that information. With the intersection of new sensing techniques and sensors that are lower in cost—not to mention more capable phone and AI and machine learning—we are at a time where this can actually start to work. We can automate a lot of the work, triage it using machine learning, and escalate the cases that look like they're emergent.

In healthcare, as with energy usage, it turns out that feedback loops are an incredibly powerful way to change people's behavior—giving them relevant information about what's happening at a time when they can actually do something about it.

Mobile phones give you both a computational platform for the interface and feedback on the device itself. At the same time, people have a huge affinity for their phones, so compliance is inherently higher. You already have this thing with you for primary reasons, so healthcare becomes a secondary use-case.

I understand that it has been an adventure to get some of this work approved by the Food and Drug Administration (FDA).

The regulatory landscape is evolving. The way the FDA looks at diagnostic tools is in terms of analytic sensitivity. If you test something using a phone, what's its absolute accuracy? But context is incredibly valuable. If I'm coughing a lot in Seattle versus in a high-risk tuberculosis region in South Africa, it means something very different, and in fact, physicians already use this context indirectly. "Are you at risk for something? Where have you been? What's your family history? What region do you reside in?" Those things aren't always built into a blood draw. So, the blood draw gives you one number, but now, machine learning can incorporate all this additional information and maybe even be more indicative of what's happening.

How have health providers reacted?

I think clinicians and clinical scientists are moving in that direction. They understand it and they see that it's where the field is heading. But health practitioners still have to think about the near term—they have to physically see patients, determine the best course of treatment, and so on. It's a challenge to bridge that gap. If you're a general practitioner who's taking care of 1,000 patients, how are you going to deal with 1,000 hemoglobin readings each day? It's just not possible, and that's why a lot of these mobile and home health technologies have not really been successful. If you can't figure out how to integrate your tools into the system we have now, the treatments are never going adapt to whatever new sensing techniques you've created.

In addition to being a professor at the University of Washington, you also spend time at Google, where you direct a health technologies group. Is there anything you can say about the work?

A lot of it is looking at new opportunities for machine learning and sensors in the healthcare space. It's still early in our explorations, but one of the exciting things about it is the opportunity to start thinking about scale. I was able to validate and prototype a lot of things in the academic world, but at Google, we can start to look at disseminating it more broadly—that's all I can say for now, but that's the high-level goal. **C**

Leah Hoffmann is a technology writer based in Piermont, NY, USA.

DOI:10.1145/3344298

Leah Hoffmann

Q&A

Inspired by the Home of the Future

2018 ACM Prize in Computing recipient Shwetak Patel pushes old tools to new heights.

SHWETAK PATEL, A professor at the University of Washington (UW), director of a health technologies group at Google, and recipient of the 2018 ACM Prize in Computing, has made a career out of pushing old tools to new heights. He has leveraged existing infrastructure to make affordable energy monitoring systems; he used mobile phone sensors like cameras and microphones to help manage chronic diseases. Here, Patel talks about feedback loops, the home of the future, and the changing healthcare landscape.

What triggered your interest in ubiquitous computing?

As an undergrad, I worked in the Georgia Tech (Georgia Institute of Technology) Aware Home, which was a facility with a bunch of different technologies that we used to explore what the home of the future would look like. We built applications for healthcare, elder care, energy monitoring, and so on, and a lot of my inspiration came from that work.

In graduate school, you began to look at how to leverage existing technologies and infrastructure to build some of those same applications in a more easily accessible way.

Sometimes, if you go straight to a specific technology, it takes a while before that can scale, but if you can take these intermediate steps where you leverage existing systems in unique ways, then you can start to answer questions about viability, usability, and effectiveness, which then informs the design of how you build it out long term.

"We built applications for healthcare, elder care, energy monitoring, and so on, and a lot of my inspiration came from that work."

One of the innovations that came out of your Ph.D. work was an energy-monitoring technique that uses a single, simple sensor deployed on the electrical system to identify what devices are drawing power. Later, at UW, you pushed that concept into new domains with technology that tracks per-fixture water use from a single sensor.

The end goal was to provide feedback for people to be able understand their energy usage and improve their mental model of where energy and water are going.

Algorithmically, what you are doing is looking at the side effects of using an appliance or a water valve. When you switch an appliance on and off, there is electrical noise, or electromagnetic interference, [CONTINUED ON P. 111]

PHOTO BY MARK STONE/UNIVERSITY OF WASHINGTON

SIGGRAPH ASIA 2019 BRISBANE

The 12th ACM SIGGRAPH Conference and Exhibition on Computer Graphics and Interactive Techniques in Asia

DREAM ZONE!

Conference 17 - 20 November 2019
Exhibition 18 - 20 November 2019

Brisbane Convention & Exhibition Centre (BCEC), Brisbane, Australia

Sponsored by:

acm

Organized by

koelnmesse

we energize your business | since 1924